Computers & Operations Research 37 (2010) 1129–1138

Contents lists available at ScienceDirect

Computers & Operations Research

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

Variable neighborhood search for the dial-a-ride problem

Sophie N. Parragh *, Karl F. Doerner, Richard F. Hartl

Department of Business Administration, University of Vienna, Bruenner Strasse 72, 1210 Vienna, Austria

ARTICLE INFO

Available online 14 October 2009

Keywords:
Transportation
Dial-a-ride problem
Metaheuristic
Variable neighborhood search

ABSTRACT

In dial-a-ride problems passengers have to be transported between pre-specified pickup and delivery locations under user inconvenience considerations. The problem variant considered in this paper aims at minimizing total routing costs while respecting maximum route duration limits, time windows, and maximum user ride time limits. We propose a competitive variable neighborhood search-based heuristic, using three classes of neighborhoods. The first neighborhood class uses simple swap operations tailored to the dial-a-ride problem; the second neighborhood class is based on the ejection chain idea; and the third neighborhood class exploits the existence of arcs where the vehicle load is zero, giving rise to natural sequences of requests. We report new best results for 16 out of 20 benchmark instances.

1. Introduction

Dial-a-ride problems (DARP) arise, e.g., in the context of patient transportation, in rural areas, or in public service for the disabled who cannot use regular public transportation systems easily. A number of transportation requests have to be served by a given fleet of vehicles. Each request is associated with a pickup and a delivery location. In contrast to the pickup and delivery problem, where goods are transported, in DARP, service oriented criteria have to be considered. Thus, usually time windows and also maximum user ride time limits have to be respected. In case user inconvenience is not taken care of by additional constraints, it is minimized in the objective function, in addition to or instead of routing costs.

Due to the application oriented character of this problem no standardized problem definition exists. The objectives applied range from the maximization of the number of patients served to the minimization of user waiting time or routing costs. The constraints considered are usually tailored to the specific problem situation. Let us give some examples. Baugh et al. [1] design a simulated annealing algorithm minimizing the weighted sum over total routing costs, time window violations (user inconvenience), and the number of vehicles used. Toth and Vigo [2], on the other hand, consider a heterogeneous fleet, service times, multiple depots, time windows and maximum user ride times, while minimizing the total distance traveled by the vehicles, using a tabu thresholding algorithm. Aldaihani and Dessouky [3] solve a mix between a DARP, i.e. transportation on demand, and a fixed route transit system by means of a tabu search heuristic. Customer inconvenience in terms of the total ride time of all the passengers and the total distance traveled by the on demand vehicles are minimized. Time windows are considered as a hard constraint. Soft time windows are considered by Jørgensen et al. [4]. They propose a genetic algorithm (GA) to solve the DARP, minimizing the weighted sum of customer transportation time, excess user ride time with respect to direct ride time, customer waiting time, time window violations, excess user ride time with respect to maximum ride time, and excess work time. This clearly shows that the DARP lacks a generic problem formulation. Additional information on existing solution methods and the problem variants considered can be found in Berbeglia et al. [5], Cordeau and Laporte [6], Parragh et al. [7]. Furthermore, we refer to Paquette et al. [8] for an overview of the different quality of service criteria that have been applied in dial-a-ride systems.

However, a rather, in our opinion, general problem definition together with a benchmark data set has been introduced by Cordeau and Laporte [9]. They consider time windows, a maximum user ride time limit, and a maximum route duration limit, while minimizing total routing costs. Results for the proposed data set obtained by means of a tabu search (TS) heuristic are reported.

We present a competitive variable neighborhood search (VNS) heuristic [10] for the same problem. An earlier version of this heuristic combined with path relinking [11] has successfully been applied in the context of the multi-objective DARP [12]. VNS has shown to be competitive when used to solve vehicle routing problems such as, among others, the multi-depot vehicle routing problem with time windows [13], the periodic vehicle routing problem [14], or the pickup and delivery traveling salesman

* Corresponding author.
E-mail addresses: Sophie.Parragh@univie.ac.at (S.N. Parragh), Karl.Doerner@univie.ac.at (K.F. Doerner), Richard.Hartl@univie.ac.at (R.F. Hartl).

0305-0548/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2009.10.003

problem with last-in-first-out loading restrictions [15]. To our best knowledge this is the first time VNS is applied to the static single-objective DARP.

The reminder of this article is organized as follows. First, a more detailed problem definition is given. This is followed by the description of the proposed metaheuristic algorithm. Finally, computational results are reported and compared to those of the TS of Cordeau and Laporte [9], and, using an adapted objective function, to the GA of Jørgensen et al. [4].

2. The dial-a-ride problem

The DARP is modeled on a complete directed graph $G=(V,A)$ where V is the set of all vertices and A the set of all arcs. For each arc (i,j) a non-negative travel cost c_{ij} and a non-negative travel time t_{ij} are considered. A total amount of n customer requests, each consisting of a pickup and delivery vertex pair $\{i, n+i\}$, are to be served by m vehicles with a capacity of Q. All vehicles are stationed at a central depot, denoted by 0 (origin depot) and $2n+1$ (end depot). Furthermore, every route has to be completed within a maximum route duration limit T. At every pickup vertex a certain number of passengers ($q_i > 0$) waits to be transported. The demand at every delivery vertex is equal to $q_{n+i} = -q_i$. Either the pickup vertex (origin) or the delivery vertex (destination) is associated with a time window $[e_i, l_i]$, depending on the type of request. Outbound requests, e.g. from home to the hospital, have a tight time window on the destination. Inbound requests, e.g. from the hospital back home, have a tight time window on the origin. The vertex that is associated with the tight time window is also referred to as the *critical vertex* of a request [9]. The service time for loading or unloading operations at each vertex is denoted by d_i. Leaving vertex i at D_i (departure time) results in arriving at the subsequent vertex j at $A_j := D_i + t_{ij}$ (arrival time). Beginning of service B_i at a vertex i cannot start before the beginning of the respective time window e_i. It is set to $B_i := \max\{A_i, e_i\}$ and vehicle waiting time to $W_i := B_i - A_i$. In order to limit user inconvenience caused by long detours, a maximum user ride time L has to be respected. The departure time from a vertex i is computed as $D_i := B_i + d_i$ and the ride time of a client corresponds to $L_i := B_{n+i} - D_i$, i.e. the time he/she spends on board the vehicle. Furthermore, also the load when leaving vertex i, denoted as y_i, has to be determined. The duration of a route k is calculated as $B^k_{2n+1} - B^k_0$, where B^k_{2n+1} (B^k_0) denotes the beginning of service at the depot $2n+1$ (0) by vehicle k. The objective is to minimize total routing costs $c(s)$, i.e. the sum over all travel costs $\sum_{(i,j)\in s} c_{ij}$ that are associated with arcs (i,j), used in a solution s. The notation used in this article is summarized in Table 1.

3. Solution framework

We solve the DARP by means of a VNS-based heuristic. In general, VNS departs from an *initial solution* s_{init}. The first incumbent solution s is equal to s_{init}. Then, in every iteration a random solution s' is generated in the current neighborhood $N_k(s)$ of s (*shaking*). A subsequent *local search* step applied to s' yields s''. If s'' is better than s, it replaces s and the search continues with the first neighborhood $k := 1$. If s'' is worse, s is not replaced and the next (larger) neighborhood is used in the subsequent iteration $k := k+1$ (*move or not*). A maximum number of neighborhoods $k_{\max}$ has to be defined. Whenever $k_{\max}$ is attained, the search continues with the first neighborhood ($k := 1$). This is repeated until some *stopping criterion* is met.

In our case, also ascending moves, fulfilling pre-specified acceptance criteria, are permitted. This means that also deteriorating solutions may become incumbent solutions with a certain probability. Furthermore, like Cordeau and Laporte [9], we allow intermediate infeasible solutions (infeasibilities are penalized). Therefore, in addition to s, we keep track of the best feasible solution s_{best} encountered so far. Also the local search step has been subject to modifications; and, in order to reduce the search space, the graph pruning and time window tightening techniques, described by Cordeau [16], are applied prior to starting the optimization procedure. In Algorithm 1 the general framework of our implementation is given. The different design elements of the VNS are described in further detail in the following paragraphs.

Table 1
Notation.

n	Number of requests
m	Number of vehicles
$V=\{0,\dots,2n+1\}$	Set of vertices
$\{i, n+i\}$	A transportation request
c_{ij}	Travel costs for arc (i,j)
t_{ij}	Travel time for arc (i,j)
Q	Vehicle capacity
T	Maximum route duration
L	Maximum user ride time
q_i	Number of passengers to be picked up at vertex i
e_i	Beginning of time window at i
l_i	End of time window at i
d_i	Service time at vertex i
A_i	Arrival time at vertex i
B_i	Beginning of service at vertex i
D_i	Departure time from vertex i
L_i	Ride time of user i
W_i	Vehicle waiting time at vertex i
y_i	Load when leaving vertex i
s	A solution (routing plan)
$c(s)$	Total routing costs of solution s

Algorithm 1. VNS

```
1:   // preprocessing
2:   do graph pruning and time window tightening;
3:   // initial solution
4:   generate s_init;
5:   set s := s_init; set k := 1;
6:   repeat
7:     // shaking
8:     randomly compute s' in N_k(s);
9:     // local search
10:    if c(s') < 1.02c(s) or p_rand < 0.01 then
11:      apply local search to s' yielding s'';
12:    else
13:      set s'' := s';
14:    // move or not
15:    if f(s'') < f(s) or meets other acceptance criteria then
16:      if c(s'') ≥ 1.05c(s) then
17:        apply local search to s'';
18:      set s := s''; set k := 0;
19:    if s'' is feasible and better than s_best then
20:      set s_best := s'';
21:    set k := (k mod k_max) + 1;
22:  until some stopping criterion is met
23:  return s_best;
```

3.1. Preprocessing

Before the optimization procedure is started, we apply graph pruning and time window tightening techniques, as described by Cordeau [16], see Algorithm 1 line 2. First, time windows are

strengthened. At all vertices that are not associated with a tight time window an artificial time window is generated. In case of an outbound request the time window at the origin i is set to $e_i = \max\{0, e_{n+i} - L - d_i\}$ and $l_i = \min\{l_{n+i} - t_{i,n+i} - d_i, H\}$, where H is the end of the planning horizon. In case of an inbound request the destination time window is set to $e_{n+i} = e_i + d_i + t_{i,n+i}$ and $l_{n+i} = \min\{l_i + d_i + L, H\}$. Then, all those arcs that cannot be part of a feasible solution are declared forbidden. These are all arcs connecting the origin depot and a destination, the destination with its origin, and an origin with the destination depot. Furthermore, request pairs are checked for compatibility regarding time windows and ride time limits. All arcs connecting incompatible parts of two requests are also declared forbidden. For the complete procedure we refer to Cordeau [16]. Forbidden arcs receive a very high value M in the distance matrix. Consequently, forbidden arcs are not likely to be used in a solution.

3.2. Initial solution

To generate the first incumbent solution s (Algorithm 1 line 4) a slightly more sophisticated procedure than employed by Cordeau and Laporte [9] is used. While Cordeau and Laporte [9] construct the initial solution completely at random, we integrate time window and spatial closeness considerations. First, requests are sorted according to some artificial beginning of service. These are set randomly within the time window of the critical vertex. Then, all routes are initialized with one request, using the first m requests on the list. After that, requests are inserted at the end of one of these partial routes in the order they appear on the list. The insertion route of the respective request is selected according to one of four minimum distance criteria. These criteria compare the distance between either origin or destination of the last request on each route and the next request on the list. Criterion one considers the two origins, criterion two the origin of the last request and the destination of the next; criterion three inspects the destination of the last and the origin of the next request; and criterion four uses the two destinations to determine the minimum distance. Which criterion is employed is selected randomly for each request individually. This is repeated until all requests have been inserted into some route. Note that a thus constructed solution might be infeasible with respect to time window and maximum route duration constraints. Finally, all routes undergo local search-based improvement (see Section 3.4). During pretests, the performance of the four distance criteria have also been evaluated separately. Usually, criterion three resulted in the best initial solution in terms of evaluation function value (see Section 3.7). However, this advantage does not translate into better results for the entire VNS. Therefore and for the purpose of generating diverse initial solutions, all four criteria are used.

3.3. Shaking

We use three different types of neighborhoods (see Algorithm 1 line 8). The first one swaps two sequences of requests, the second applies the idea of ejection chains [17], and the third redistributes requests forming a "natural" sequence to different routes.

Swap neighborhood (S): In the first neighborhood class two sequences of vertices are exchanged. First, two routes are chosen randomly. Then, on each route a sequence to be swapped is selected: first, the starting vertices of the two sequences are randomly selected. Then, the length of each sequence is randomly chosen. The maximum sequence length is referred to as the size of this neighborhood. Finally, all requests forming the respective sequences are deleted from their original routes and inserted one-by-one into the other route. Each request is inserted as follows: first, the critical vertex of the respective request (i.e. the one with the tight time window) is inserted at its best position; then, the non-critical vertex is inserted at its best position in accordance with the critical one. A swap of size three, e.g., consists in exchanging two sequences of at most length three.

Note that for each vertex within the selected sequence the corresponding origin or destination has to be moved as well, even if it is not part of the sequence. This is depicted in Fig. 1(a). Let i^+ denote the origin and i^- the destination of request i. Here, both vertices forming request 6 are part of the selected sequence. In case of request 2, only the destination is part of the selected sequence. However, also its origin, denoted by 2^+, will be moved.

Chain neighborhood (C): The second neighborhood class applies the ejection chain idea [17]. In contrast to its original version, the number of routes affected is a parameter. In a first step, two routes are chosen. From the first route a sequence of requests (selected as in the swap neighborhood) is moved to the second route. In a second step, from the second route the sequence that decreases the evaluation function value (see Section 3.7) of this route the most is moved to a third route (it may also be the first route). The second step is repeated until the maximum number of sequences moved has been reached. All insertions of sequences into their new routes are done one-by-one in the best possible way, using the notion of critical vertices. All routes are selected randomly. The neighborhood size in this context refers to the maximum number of sequences moved and the maximum length of a sequence to be selected within each route. E.g., a chain neighborhood of size three consists of moving a random sequence of at most three requests from its original route to a second route. From the second route the sequence of at most three requests that decreases the route's evaluation function value the most is moved to a third route. Finally, from this third route again the maximum savings sequence with a length of at most three is moved to a fourth route. Thus, three sequences are moved and three plus one routes are affected.

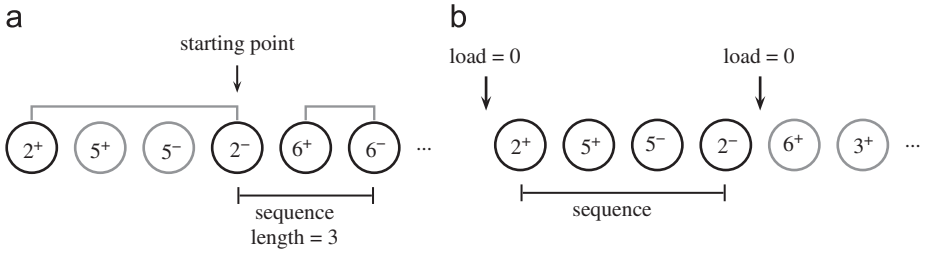

Fig. 1. Different sequences. (a) Swap sequence. (b) Zero split sequence.

Zero split neighborhood (Z): The third neighborhood is a parameterless neighborhood. As in the two previous neighborhoods, a sequence of requests is moved. However, this sequence is chosen in a different way. Within every route "natural" sequences of requests can be defined. These sequences lie between two arcs where the vehicle load is equal to zero, i.e. the vehicle is empty (see Fig. 1(b)). Every route contains at least two such arcs. These are the arcs from the depot to the first origin along the route and from the last destination back to the depot. In addition, we discovered that routes quite often contain more than only one sequence of this type. This led to the idea of the zero split neighborhood. In this neighborhood a random sequence of these natural sequences is removed from a random route. All requests part of this sequence are then reinserted one-by-one into different routes chosen at random. Insertion is again done one-by-one in the best possible way using the notion of critical vertices.

Neighborhood sequence: The neighborhoods derived from these three neighborhood classes are applied in the following order (the number given in addition to the neighborhood abbreviation indicates the neighborhood size): S1–C1–S2–C2–S3–C3–S4–C4–S5–C5–S6–C6–Z. This means that $N_1 = \mathrm{S1}$, $N_2 = \mathrm{C1}$, etc. (see Algorithm 1 line 8). Thus, a total of $k_{\max} = 13$ different neighborhoods but only three neighborhood classes are used. The idea of the VNS is to increase the size of the applied neighborhood from one iteration to the other. To mimic this behavior the above sequence was chosen: S is ordered before C because in C more routes and also more requests may be affected. E.g., in S2 at most four requests (two sequences of length two) are moved and two routes are affected, in C2 also at most four requests are moved but at most three routes are affected.

3.4. Local search

While in the shaking step mainly inter-tour neighborhood operators are applied, the local search step of the proposed VNS (Algorithm 1 line 11) is of the intra-tour first improvement type. It moves along each route as follows. In a first step, the first origin and its corresponding destination are removed from the route. Then, the critical vertex of the two is re-inserted at the first possible position with respect to time window feasibility. Thereafter, the noncritical vertex is inserted at the first possible position with respect to the critical vertex. In this case, the first possible position is the one closest to the critical vertex: if the critical vertex is an origin, the corresponding destination is inserted right after the critical vertex. If the critical vertex is a destination, the corresponding origin is inserted directly before the critical vertex. If this positioning improves the route's evaluation function value, the two vertices are kept at their new positions and the search continues with the new first origin on the route. Otherwise, the noncritical vertex is tentatively inserted at the next possible position, increasing the distance to the critical one. In case of an improvement the search is restarted. Otherwise, the noncritical vertex is moved to the next possible position. This is repeated until there is no other insertion position for the noncritical vertex possible. In this case, the critical vertex is moved to the next possible position, and the noncritical vertex is again inserted in accordance with the critical one. This is repeated until either an improved positioning is encountered or the last possible positioning has been reached. In this case, the vertex pair is kept at its original position, and the next origin on the route is searched. The procedure ends as soon as the last origin on the route and its destination have been tried at all possible positions and no improvement has been found.

In preliminary tests, instead of the above described procedure the much simpler and therefore also faster route improvement method used by Cordeau and Laporte [9] has been employed. The results showed, however, that the above described procedure works better than our implementation of Cordeau and Laporte's method.

3.5. Local search frequency

In standard VNS implementations, the local search step is conducted after every shaking step. However, in our case, we always insert the requests to be moved or swapped one-by-one in the best possible way (best insertion), using the notion of critical vertices, as defined by Cordeau and Laporte [9]. Thus, request insertion already includes some kind of local search for the inserted request. Consequently, the local search step is not absolutely necessary.

In order to avoid unnecessary calls to the time consuming local search heuristic, we have chosen to use local search only if s' is a *promising* solution. A *promising* solution is a solution that has some potential to become a new incumbent solution. After several experiments, we defined as *promising* a solution s' that is at most 2% worse than the incumbent s. Since the objective of the DARP is to minimize total routing costs $c(s)$, a promising solution is characterized by $c(s') < 1.02c(s)$. In order to introduce another element of diversification, every solution has a 1% chance to be subject to local search-based improvement, see Algorithm 1 line 9. The reason for choosing $c(s)$ instead of the evaluation function value, which is explained in detail in Section 3.7, refers to the fact that in the case where some constraint violation is penalized severely, already small violations will yield very high evaluation function values; although small constraint violations are very likely to be repaired in the local search step. Thus, small constraint violations in combination with a high penalty factor would be treated in the same way as large constraint violations in combination with a low penalty factor. In order to circumvent this problem, the promising solution property is solely defined on $c(s)$.

Furthermore, every solution s'' which meets the acceptance criteria and will thus become the new incumbent solution is also subject to local search-based improvement; given that its routing costs are at least 5% worse than those of the current incumbent, see Algorithm 1 line 16.

Surprisingly, preliminary tests showed that a 5% limit is better than a limit of 2%. A limit of 2% would mean that every solution constituting a new incumbent solution undergoes local search; either due to its *promising* solution property or because it will be the new incumbent solution. A possible reason why the 5% limit worked better than the 2% one is increased diversification: a solution that has not been locally optimized may, in some cases, provide better options regarding the removal and reinsertion of a request in the subsequent iteration than a locally optimized one.

3.6. Move or not

In order to decide whether the search should move to the new solution s'' or not (see Algorithm 1 line 14), i.e. if s'' replaces s, as in Hemmelmayr et al. [14] and Ropke and Pisinger [18] a simulated annealing [19] type criterion is used. Let $f(s)$ denote the evaluation function value of solution s (see Section 3.7). In the beginning s'' can only replace s if $f(s'') < f(s)$. As soon as the first feasible solution has been found, also deteriorating solutions may become incumbent solutions with probability $\exp(-[f(s'') - f(s_{best})]/t)$. The initial temperature is then set such that if $[f(s'')/f(s_{best})] - 1 = 0.005$, s'' is accepted with a probability of 0.2. Thereafter, t is linearly decreased until 0, i.e. when the maximum number of iterations

has been attained $t=0$. The above values have been determined based on pretests on a smaller set of instances.

3.7. Solution evaluation

Following Cordeau and Laporte [9] we penalize load violation $q(s)$, duration violation $d(s)$, time window violation $w(s)$, and ride time violation $t(s)$ in the evaluation function. Load violation is computed as $q(s)=\sum_{i=1}^{2n}(y_i-Q)^+$, where $x^+=\max\{0,x\}$, duration violation as $d(s)=\sum_{k=1}^{m}(B_{2n+1}^k-B_0^k-T)^+$, time window violation as $w(s)=\sum_{i=1}^{2n}(B_i-l_i)^+$, and ride time violation as $t(s)=\sum_{i=1}^{n}(L_i-L)^+$. For the corresponding notation see Section 2. Let now $c(s)$ denote the total routing costs of all vehicles, which is the sum of the costs c_{ij} associated with the arcs (i,j) traversed by the vehicles, the following evaluation function is applied:

$$f(s)=c(s)+\alpha q(s)+\beta d(s)+\gamma w(s)+\tau t(s). \quad (1)$$

The penalty terms for load, duration, time window, and ride time violation are given by α, β, γ, and τ, respectively. These are set to $\alpha=\beta=\gamma=\tau=1$ in the beginning of the search. Similar to Cordeau and Laporte [9], their values are then adjusted in a dynamic way. Every time a new incumbent solution s is identified, the penalty parameters are either increased or decreased. In case s violates a penalized constraint, the corresponding penalty term is multiplied by δ. If, e.g., s violates a capacity constraint $q(s)>0$, then $\alpha:=\alpha(1+\delta)$. On the other hand, if $q(s)=0$, then $\alpha:=\alpha/(1+\delta)$. The value of δ is randomly chosen between $\underline{\delta}=0.05$ and $\overline{\delta}=0.1$, every time a new incumbent is identified. Using different values for δ decreases the chances of cycling, and works as a diversification mechanism. Note that a solution s'' can only become s_{best} if and only if $q(s'')=d(s'')=w(s'')=t(s'')=0$, see Algorithm 1 line 19.

In order to set the beginning of service in the best possible way, such that route duration is minimum and ride time limits are respected where possible, the route evaluation scheme introduced by Cordeau and Laporte [9] is applied. It is based on the forward time slack F_i, defined by Savelsbergh [20], adjusted to the DARP,

$$F_i=\min_{i\le j\le q}\left\{\sum_{i<p\le j}W_p+(\min\{l_j-B_j,L-P_j\})^+\right\}, \quad (2)$$

where W_p denotes the waiting time at vertex p, q the last vertex on the route, and P_j the ride time of the user whose destination is $j\in\{n+1,\ldots,2n\}$ given that $j-n$ is visited before i on the route; $P_j=0$ for all other j. The slack at vertex j is the cumulative waiting time until j, plus the minimum of the difference between the end of the time window and the beginning of service at j, and the difference between the maximum user ride time and P_j. The forward time slack at vertex i is the minimum of all slacks between i and q. F_0 thus gives the maximum amount of time by which the departure from the origin depot can be delayed (with equal time window violations and smaller or equal ride time violations) to yield a modified route of minimal duration. Furthermore, Cordeau and Laporte [9] observed that delaying the departure by $\sum_{0<p<q}W_p$ does not affect the arrival time at q, whereas delaying the departure by more will only increase the arrival time at q by as much. Consequently, the departure from the depot should only be delayed by at most $\min\{F_0,\sum_{0<p<q}W_p\}$.

The forward time slack can also be applied to delay B_i at each origin vertex such that the ride time of the corresponding request is reduced. This reasoning led Cordeau and Laporte [9] to an eight-step evaluation scheme. We use the same eight-step scheme. However, in case of no more reparable violations or irreparable violations (e.g., time window violations), we stop and use the current approximated violations in the evaluation function. The whole eight-step scheme is given in Algorithm 2.

Algorithm 2. Eight-step evaluation scheme.

1. Set $D_0:=e_0$.
2. Compute A_i, W_i, B_i, D_i and y_i for each vertex i on the route.
   If some $B_i>l_i$ or $y_i>Q$ GO TO STEP 8.
3. Compute F_0.
4. Set $D_0:=e_0+\min\{F_0,\sum_{0<p<q}W_p\}$.
5. Update A_i, W_i, B_i and D_i for each vertex i on the route.
6. Compute L_i for each request on the route.
   If all $L_i\le L$ GO TO STEP 8.
7. For every vertex j that is an origin
   (a) Compute F_j.
   (b) Set $W_j:=W_j+\min\{F_j,\sum_{j<p<q}W_p\}$ $B_j:=A_j+W_j$; $D_j:=B_j+d_j$
   (c) Update A_i, W_i, B_i and D_i for each vertex i that comes after j in the route.
   (d) Update L_i for each request i whose destination is after j.
   If all $L_i\le L$ of requests whose destinations lie after j
   GO TO STEP 8.
8. Compute changes in violations of vehicle load, duration, time window and ride time constraints.

The eight-step procedure first minimizes time window constraint violations in steps (1) and (2). In addition to the original version, also the vehicle load at each vertex on the route is calculated. In case either of the two (time window or loading restriction) is violated, an irreparable violation is encountered and we proceed with step (8). Time window violations cannot be repaired hereafter since all B_i are set to the earliest feasible point in time, which is, coming from vertex j, $D_j:=(B_j+d_j)$, $A_i:=(D_j+t_{ji})$, $W_i:=(e_i-A_i)^+$, and $B_i:=(A_i+W_i)$. In steps (3)–(6) route duration is minimized, without increasing time window violations. Here, a ride time feasibility step is inserted. In case all ride times are already feasible, we can skip step (7) and go directly to step (8). In step (7) ride times are sequentially minimized. Here again the feasibility of the ride times of those requests whose destinations lie behind j is checked. In case all of them are feasible, go to step (8). In case the current route contains a forbidden arc, the eight-step scheme is not used. Instead, only the routing costs (including the costs of the forbidden arcs) and the different load levels along the route are calculated. Duration, time window and ride time constraint violations are set to M multiplied by the number of forbidden arcs part of the route.

3.8. Stopping criterion

Initially the stopping criterion of the proposed VNS was set to a limit on the maximum number of iterations. One iteration corresponds to constructing one new solution (shaking) and to optimize it locally in the local search step. However, especially in case of small problem instances, the best (possibly optimal) solution is usually found very quickly. Therefore, a second stopping criterion has been defined. This criterion is the maximum number of iterations between the identification of two new global best solutions s_{best}. If during 2×10^6 iterations no new global best solution can be generated, the search terminates and returns s_{best}. Summarizing, the search either stops as soon as the maximum total number of iterations (10^7) has been attained, or if during 2×10^6 iterations no new global best solution can be identified.

3.9. Delineation from earlier version

An earlier version of the proposed algorithm, using similar neighborhood operators, was developed to solve the bi-objective DARP in combination with path relinking. In Parragh et al. [12] a

numerical analysis of this version was presented. In the current paper we now focus on a cost-oriented single objective formulation from the literature. Furthermore we show that the evaluation mechanism can easily be adapted to deal with a weighted sum objective function, consisting of many different components. Based on extensive tests and parameter tuning, we provide further refinements of the algorithm tailored to the characteristics of these single objective problems. The major differences with respect to the earlier version correspond to the new initial solution construction heuristic, the faster evaluation mechanism, the reduced number of neighborhood classes, the new local search frequency (depending on solution quality), and the additional stopping criterion. Especially, the newly defined local search frequency and the more efficient evaluation function improve run times considerably and thus permit the exploration of larger neighborhoods. In Parragh et al. [12] the maximum neighborhood size within a neighborhood class was four, i.e. the largest neighborhoods were a move, a swap, and a chain neighborhood of size four. In the current paper the maximum neighborhood size per operator is six. Using this limit, the move neighborhood class is no longer necessary in order to obtain high quality solutions. Summarizing, these refinements have increased our method's simplicity as well as its efficiency.

4. Computational results

The VNS was implemented in C++. All experiments were conducted on a Pentium D computer with 3.2 GHz. Our algorithm was tested on the benchmark data set of Cordeau and Laporte [9]. Based on this data set, two different versions of the DARP have been considered in the literature.

In the standard version of the DARP, as introduced in Section 2, all constraints are hard constraints and the objective is the minimization of total routing costs, which corresponds to minimizing the total travel distance; since $c_{ij} = t_{ij}$ for all i and j. This version was also considered by Cordeau and Laporte [9]. We compare the results of our VNS with the best results obtained by the TS developed by the same authors.

In the modified version introduced by Jørgensen et al. [4] a different objective function was selected, namely a weighted combination of routing costs, total route duration, user ride time, user waiting time, and penalties for violations of route duration, time window, and ride time is minimized. All time related constraints are thus considered to be soft.

In the following, first the test data set is described. Then, results obtained by the VNS are compared to those of the TS by Cordeau and Laporte [9]. Finally, results for the modified version and comparisons to the GA by Jørgensen et al. [4] are presented.

4.1. Test instances

Cordeau and Laporte [9] proposed a data set of 20 randomly generated instances, containing between 24 and 144 requests. In each instance the first $n/2$ requests were defined as inbound while the remaining $n/2$ were defined as outbound requests. Origin and destination locations were generated using the procedure introduced in Cordeau et al. [21]. For each vertex a service time $d_i = 10$ was set. The load was set to $q_i = 1$ if $i \in \{1, \ldots, n\}$ and to $q_i = -1$ if $i \in \{n+1, \ldots, 2n\}$. Routing costs c_{ij} and travel times t_{ij} from a vertex i to a vertex j are equal to the Euclidean distance between these two vertices. Each vertex is associated with a time window $[e_i, l_i]$. Origins of outbound requests and destinations of inbound request do not have a tight time window. Thus, their time windows were set to the length of the planning horizon $[0, 1440]$. The data set was split into two groups. In group (a) narrow time windows were set while in group (b) wider time windows were created. For all instances route duration was set to $T = 480$, vehicle capacity to $Q = 6$, and maximum user ride time to $L = 90$. Furthermore, for instances R1a–R6a and R1b–R6b the number of vehicles was set such that routes are only moderately full; instances R7a–R10a and R7b–R10b might be infeasible with fewer vehicles.

4.2. Comparison to tabu search on standard DARP

To our best knowledge no other solution method, besides the TS, has been used to solve this set of instances, using the minimum distance objective function.

In order to illustrate the performance of the swap and the chain neighborhood classes within our VNS framework, they were tested separately on a limited number of instances. For testing purposes, four instances of intermediate size were selected from each instance subset: R3a, R8a, R3b, and R8b. Five random runs were performed for each of these instances, using as sole stopping criterion a limit of 10^6 iterations. In case of the swap operator, we used neighborhoods S1–S2–S3–S4–S5–S6, in case of the chain operator, C1–C2–C3–C4–C5–C6. Taking average values over five random runs, using neighborhoods S1–…–S6, an average deviation of 2.06% from the best known solutions by Cordeau and Laporte [9] was obtained. Taking average values over five random runs, using neighborhoods C1–…–C6, an average deviation of 1.41% was achieved. However, as expected, the run times differed considerably: setting S1–…–S6 consumed, on average, less than half of the run time (7.42 min) when compared to the average run time of setting C1–…–C6 (18.38 min). Combining these two neighborhood classes as described in Section 3.3 (S1−C1−S2−C2 ⋯ S6−C6) results in an average deviation of 1.20% with respect to the best known solution values and in an average run time of 14.71 min. This supports the combination of the two neighborhood classes: solutions of higher quality are computed within intermediate run times. Ordering the smallest swap neighborhood before the smallest chain neighborhood and continuing the neighborhood sequence in this way is justified by the shorter run times of the swap operator.

In a next step, the VNS (using both neighborhood classes) was applied to the whole data set. Despite the promising results in the previous tests, for one instance and one random seed, no feasible solution could be identified. This problem was resolved by the introduction of the zero split neighborhood operator, ordered last in the neighborhood sequel. In Table 2, the results thus obtained within 10^6 iterations are summarized and compared to the results of Cordeau and Laporte's TS, run for 10^4 iterations. Column one contains the name of the instance; columns two and three the size in terms of the number of available vehicles (m) and the number of requests (n). Then, the solution values and the according CPU times in minutes obtained by means of one run of the TS are provided. This is followed by average values for two runs of the TS and the best values out of these two runs. Column "TS (10 runs)" provides the best values out of 10 runs of the TS by Cordeau and Laporte [9]. Several runs of their TS yield different solutions since the initial solution is constructed randomly; and also the tabu tenure as well as the amount of adjustment, regarding the penalty terms, includes some randomness. Unfortunately average values over these 10 runs are not provided in Cordeau and Laporte [9]. The subsequent columns present average values for five random runs of the proposed VNS within 10^6 iterations, average CPU times in minutes and the best values out of these five random runs. All entries marked in bold correspond to the best values per instance identified across all solution values reported for the according instance in this table.

Table 2
VNS (10^6 iterations) vs. TS (10^4 iterations).

	m	n	TS (1 run)		TS (2 runs)		TS (10 runs)	VNS (5 runs)		
				CPU[a]	Avg.	Best	Best	Avg.	CPU[b]	Best
R1a	3	24	190.79	1.90	190.79	190.79	**190.02**	**190.02**	4.01	**190.02**
R2a	5	48	303.60	8.06	303.74	303.60	302.44	304.55	6.72	**302.39**
R3a	7	72	541.25	17.18	538.43	535.60	**534.95**	540.84	7.84	535.94
R4a	9	96	596.94	28.77	592.45	587.95	**572.78**	588.27	11.95	579.83
R5a	11	120	663.10	46.24	657.92	652.73	644.15	648.34	20.14	**638.63**
R6a	13	144	823.52	53.87	825.99	823.52	**803.20**	821.87	24.75	810.52
R7a	4	36	294.77	4.39	293.79	292.80	**291.71**	294.95	4.44	**291.71**
R8a	6	72	495.71	20.44	496.67	495.71	**494.89**	497.40	11.60	496.35
R9a	8	108	683.15	50.51	686.52	683.15	678.09	679.09	31.42	**670.55**
R10a	10	144	884.10	87.53	889.42	884.10	880.25	901.23	45.52	**875.45**
R1b	3	24	164.98	1.93	164.85	164.72	164.58	165.00	5.10	**164.46**
R2b	5	48	302.91	8.29	302.10	301.28	299.55	301.88	9.65	**299.40**
R3b	7	72	501.17	18.54	499.69	498.20	493.30	496.48	14.72	**491.43**
R4b	9	96	557.93	31.18	553.41	548.89	**535.90**	548.31	24.46	543.28
R5b	11	120	591.60	54.33	592.13	591.60	**591.60**	600.21	35.98	592.40
R6b	13	144	775.24	73.70	770.90	766.55	**754.71**	766.34	42.78	755.47
R7b	4	36	250.88	4.23	249.67	248.46	248.47	249.38	5.52	**248.21**
R8b	6	72	472.69	22.86	472.00	471.31	**462.69**	474.39	17.33	463.03
R9b	8	108	607.85	51.28	609.64	607.85	**607.85**	616.98	31.60	612.74
R10b	10	144	831.10	92.41	825.64	820.18	820.16	830.63	75.32	**815.09**
Avg.			526.66	33.88	525.78	523.45	518.56	525.81	21.54	518.85

Avg=average.

[a] Run time in minutes on a Pentium 4 computer with 2 GHz.
[b] Average run times in minutes on a Pentium D computer with 3.2 GHz.

Table 3
Deviations—VNS (10^6 iterations) vs. TS (10^4 iterations).

	VNS (5 runs, avg.) vs.		VNS (5 runs, best) vs.	
	TS (1 run)	TS (2 runs, avg.)	TS (2 runs, best)	TS (10 runs, best)
R1a	−0.40	−0.40	−0.40	0.00
R2a	0.38	0.33	−0.40	−0.02
R3a	−0.08	0.45	0.06	0.19
R4a	−1.45	−0.70	−1.38	1.23
R5a	−2.23	−1.45	−2.16	−0.86
R6a	−0.20	−0.50	−1.58	0.91
R7a	0.06	0.40	−0.37	0.00
R8a	0.34	0.15	0.13	0.30
R9a	−0.60	−1.08	−1.84	−1.11
R10a	1.94	1.33	−0.98	−0.55
R1b	0.01	0.09	−0.16	−0.07
R2b	−0.34	−0.07	−0.62	−0.05
R3b	−0.94	−0.64	−1.36	−0.38
R4b	−1.72	−0.92	−1.02	1.38
R5b	1.46	1.37	0.14	0.14
R6b	−1.15	−0.59	−1.45	0.10
R7b	−0.60	−0.12	−0.10	−0.10
R8b	0.36	0.51	−1.76	0.07
R9b	1.50	1.20	0.80	0.80
R10b	−0.06	0.60	−0.62	−0.62
Avg.	−0.19	−0.01	−0.75	0.07

Avg=average.

In Table 3 the deviations of the proposed VNS, run for 10^6 iterations, with respect to the TS of Cordeau and Laporte [9], run for 10^4 iterations, are summarized. Comparing average values over five runs of the VNS to one run of the TS, the VNS is on average 0.19% better. Comparing them to average values over two runs of the TS, results of almost equal quality are obtained by the two procedures. Comparing the best values out of five random runs of the VNS to the best values out of the two random runs of the TS, the proposed VNS is on average 0.75% better. Comparing them to the best out of 10 random runs of the TS, the TS obtains slightly better results (0.07%). Summarizing these results, the two methods achieve results of comparable quality.

In order to further illustrate the effectiveness of the three neighborhood classes, we kept track of the number of times each of the neighborhood operators contributed a new incumbent solution. Table 4 contains this information in terms of average values over five random runs for each instance and each neighborhood within 10^6 iterations. It clearly shows that all neighborhoods contribute to the search. Especially remarkable is the fact that the zero split neighborhood, although ordered last, still contributes a considerable number of new incumbent solutions. In case of the smaller neighborhoods (S1, C1, S2, C2, S3, C3, S4, and C4), the chain operator contributes more incumbent solutions than the swap operator. This relation changes for S5, C5, S6, and C6. The chosen ordering is, however, still justifiable by the shorter run times of the swap-based operators. Thus, in all those cases where the two neighborhoods have a similar catchment area, the one with the lower run time is ordered first.

Finally, Table 5 gives the results obtained with the proposed VNS, using the stopping criterion defined in Section 3.8, compared to the best known solutions obtained by the TS. Column one contains the name of the instance. Column "TS" gives the best solutions of the TS described in Cordeau and Laporte [9]. These solutions are the best solutions across all experiments conducted by the previous mentioned authors, i.e. either the best out of 10 runs of the TS with 10^4 or one run with 10^5 iterations. The subsequent columns give the results obtained by means of the VNS proposed in this paper. First, average values over five independent runs and the deviations from the best known solutions are given. Negative percentage deviations indicate an improved solution with respect to the best known value. Those solution values are given in bold. The subsequent columns give the best solution values out of these five random runs and the respective deviations from the best known values. Column "CPU" gives the average run time in minutes per instance and column "iterations" provides the average number of iterations used. The

Table 4
Neighborhood usage VNS (10^6 iterations, average values over 5 runs).

	Number of times new incumbent found by												
	S1	C1	S2	C2	S3	C3	S4	C4	S5	C5	S6	C6	Z
R1a	34.6	43.0	19.2	36.2	13.2	30.2	7.8	21.0	5.4	14.8	6.2	12.4	17.6
R2a	169.8	257.6	78.4	195.4	44.2	98.6	32.0	45.6	25.4	19.2	23.6	16.4	116.4
R3a	249.4	435.2	134.2	268.4	67.6	142.2	43.6	73.4	39.4	37.0	40.6	24.2	155.4
R4a	480.4	719.6	229.0	492.0	138.2	259.2	99.2	131.4	90.2	79.8	79.2	41.2	266.4
R5a	728.6	1107.4	362.2	758.6	213.8	371.2	154.4	174.6	124.0	97.2	119.4	60.6	507.4
R6a	1039.6	1589.2	521.8	1010.2	320.0	478.4	217.8	229.0	178.2	127.4	151.4	75.8	599.2
R7a	148.2	230.2	75.2	174.6	42.4	112.4	31.6	58.8	25.2	29.8	22.6	20.4	114.2
R8a	192.8	305.8	80.8	258.8	52.0	142.4	30.6	63.8	25.4	33.2	27.0	20.2	117.6
R9a	449.6	670.0	221.8	459.6	117.4	244.0	83.6	112.4	58.8	58.8	47.0	37.4	340.4
R10a	791.0	1146.4	376.8	723.8	200.0	306.4	140.0	138.6	98.6	66.4	76.8	38.8	231.4
R1b	24.6	33.4	12.8	48.8	7.6	29.6	7.0	15.8	6.8	9.4	4.4	6.8	38.0
R2b	125.6	223.8	62.4	171.6	36.4	86.2	29.6	53.6	27.4	32.4	26.2	21.6	145.2
R3b	276.2	436.2	121.2	314.6	74.4	179.4	54.4	94.6	44.6	46.0	43.8	30.8	194.8
R4b	506.2	709.0	246.2	499.0	148.0	249.8	103.6	108.4	80.0	61.6	70.0	43.0	274.6
R5b	754.4	1242.6	384.8	786.8	225.4	400.6	164.0	204.6	142.4	110.2	132.0	68.8	496.8
R6b	1088.4	1569.4	556.2	1012.4	332.8	516.0	243.8	260.2	193.8	135.8	183.4	85.8	717.0
R7b	66.2	103.0	33.8	92.0	14.2	63.8	13.4	25.8	5.8	14.6	7.0	7.4	35.4
R8b	233.8	366.8	103.8	260.6	51.8	130.6	32.8	63.2	29.0	33.2	25.8	16.8	122.8
R9b	485.8	723.8	213.0	499.4	118.2	223.8	77.0	109.8	57.6	46.6	41.2	33.4	224.0
R10b	944.8	1399.8	463.6	857.4	252.6	400.0	164.8	184.4	121.8	88.2	93.6	59.8	754.6
Avg.	439.5	665.6	214.9	446.0	123.5	223.2	86.6	108.5	69.0	57.1	61.1	36.1	273.5

Avg = average.

Table 5
VNS vs. TS.

	m	*n*	TS[a]	VNS (5 runs)						All tests	
				Avg.	(%)	Best	(%)	CPU[b]	Iterations	Best	(%)
R1a	3	24	**190.02**	**190.02**	0.00	**190.02**	0.00	8.98	2005 471.4	**190.02**	0.00
R2a	5	48	302.08	**301.78**	−0.10	**301.34**	−0.24	20.02	3090 374.2	**301.34**	−0.24
R3a	7	72	532.08	536.08	0.75	533.01	0.17	33.21	4550 849.6	**532.00**	−0.02
R4a	9	96	572.78	578.21	0.95	573.92	0.20	63.73	5537 700.6	**570.25**	−0.44
R5a	11	120	636.97	637.72	0.12	**636.77**	−0.03	154.47	8807 576.0	**628.11**	−1.39
R6a	13	144	801.40	809.27	0.98	802.12	0.09	230.06	9626 121.4	**794.06**	−0.92
R7a	4	36	**291.71**	294.26	0.87	**291.71**	0.00	9.62	2210 062.8	**291.71**	0.00
R8a	6	72	494.89	495.70	0.16	**490.58**	−0.87	52.94	4329 319.8	**487.84**	−1.42
R9a	8	108	672.44	673.18	0.11	**666.96**	−0.81	143.08	5271 912.8	**658.31**	−2.10
R10a	10	144	878.76	**873.15**	−0.64	**866.97**	−1.34	354.97	8861 509.0	**857.11**	−2.46
R1b	3	24	**164.46**	**164.46**	0.00	**164.46**	0.00	12.87	2450 143.4	**164.46**	0.00
R2b	5	48	296.06	299.19	1.06	296.65	0.20	25.89	2770 660.4	**295.66**	−0.14
R3b	7	72	493.30	494.23	0.19	**490.16**	−0.64	44.32	3123 281.0	**486.57**	−1.36
R4b	9	96	535.90	540.50	0.86	**533.15**	−0.51	127.43	6160 531.6	**530.70**	−0.97
R5b	11	120	589.74	**588.48**	−0.21	**583.12**	−1.12	273.99	7994 202.4	**578.61**	−1.89
R6b	13	144	743.60	751.55	1.07	**742.28**	−0.18	341.00	8925 070.4	**740.35**	−0.44
R7b	4	36	**248.21**	**248.21**	0.00	**248.21**	0.00	16.81	2715 830.8	**248.21**	0.00
R8b	6	72	462.69	468.97	1.36	**462.50**	−0.04	56.96	3510 304.2	**461.39**	−0.28
R9b	8	108	601.96	607.94	0.99	**600.18**	−0.30	152.58	5892 029.8	**597.75**	−0.70
R10b	10	144	798.63	805.93	0.91	**796.90**	−0.22	543.05	9358 576.2	**795.16**	−0.43
Avg.			515.38	517.94	0.47	513.55	−0.28	133.30	5359 576.4	510.48	−0.76

Avg = average.

[a] Best known solutions computed by Cordeau and Laporte [9].
[b] Average run times in minutes on a Pentium D computer with 3.2 GHz.

last two columns provide the best solution values obtained during all experiments in the parameter tuning phase of the VNS and their respective deviations from the best known values. Cordeau and Laporte [9] obtain their best results with an average run time of approximately 338.82 min on a Pentium 4, 2 GHz computer, for either 10 random runs with 10^4 iterations or one run with 10^5 iterations. The proposed VNS needs on average 133.30 min. Taking average values over five runs, it is on average less than 0.5% worse than the TS. Taking the best values out of these five runs, it improves the solutions of the TS by, on average, 0.28%. In addition, the given average number of iterations traversed during the search clearly indicates that the additional stopping criterion, i.e. a limit on the number of non-improving iterations, is justified; on average, about 5×10^6 iterations are sufficient to yield solutions of high quality. Finally, taking the best values across all versions tested in the parameter tuning phase, including experiments with an additional move-based neighborhood operator (see [22, Chapter 3]), the proposed VNS finds 16 new best solutions and

four ties. Summarizing these results, the proposed VNS is able to generate high quality solutions within reasonable computation times.

4.3. Comparison to genetic algorithm with modified objective function

In order to test the flexibility of the proposed VNS, we adapted the objective function to the one used by Jørgensen et al. [4]. As stated above, they minimize a weighted combination of total routing costs, total excess ride time with respect to direct ride time, total waiting time with passengers aboard the vehicle, and route duration. Furthermore, the solution framework of Jørgensen et al. [4] allows time window, route duration, and ride time violations, but penalizes them. These constraints are therefore treated as soft constraints. Only vehicle capacity is treated as a hard constraint. The objective function thus applied is the following:

$$f'(s) = w_1 c(s) + w_2 r(s) + w_3 l(s) + w_4 g(s) + w_5[w(s) + e(s)] + w_6 t(s) + w_7 d(s). \quad (3)$$

The term $r(s)$ denotes excess ride times with respect to direct ride times and is computed as $r(s) = \sum_{i=1}^{n}(B_{n+i} - D_i - t_{i,n+i})$; $l(s)$ denotes the sum over all waiting times weighted by the number of passengers aboard the vehicle when waiting, $l(s) = \sum_{i=1}^{2n}[W_i(y_i - q_i)]$; $g(s)$ gives the sum over all individual route durations, $g(s) = \sum_{k=1}^{m}(B^k_{2n+1} - B^k_0)$; $e(s)$, finally, denotes the sum over early arrivals, $e(s) = \sum_{i=1}^{2n}(e_i - A_i)^+$. Early arrivals are penalized in the same way as late arrivals $w(s)$. Jørgensen et al. [4] set the weights to $w_1 = 8$, $w_2 = 3$, $w_3 = 1$ and $w_4 = 1$, and $w_5 = w_6 = w_7 = n$.

In order to accommodate the above objectives, we adapted our evaluation function in the following way:

$$f''(s) = w_1 c(s) + w_2 r(s) + w_3 l(s) + w_4 g(s) + \alpha q(s) + \beta d(s) + \gamma w(s) + \tau t(s). \quad (4)$$

We thus minimize routing costs $c(s)$, excess ride time $r(s)$, waiting with passengers aboard $l(s)$, and route duration $g(s)$, using the same weights as Jørgensen et al. [4]; and we penalize load violation $q(s)$, duration violation $d(s)$, time window violation $w(s)$, and ride time violation $t(s)$. Note that, as before, in our framework a solution s can only become a new global best solution s_{best}, if and only if $q(s) = d(s) = w(s) = t(s) = 0$. Our framework thus still treats vehicle capacity, maximum route duration, maximum ride time, and time windows as hard constraints. Early arrivals $e(s)$ are neither penalized nor prohibited in our case. We still assume that a vehicle can arrive early at a pickup or delivery location but it will have to wait until service is possible. The waiting time for passengers, which incurs due to this assumption, will be minimized by the term $l(s)$ in the objective function.

Jørgensen et al. [4] run their algorithm five times and provide average values over these five runs for total route duration $g(s)$, total and average vehicle waiting time, and total and average ride time. The according objective function values $f'(s)$ as well as the values for the total distance traveled $c(s)$ and passenger waiting time $l(s)$, see Table 6, are taken from Bergvinsdottir [23]. Total vehicle waiting time is computed as $\sum_{i=1}^{2n} W_i$ and average ride time as $\sum_{i=1}^{n} L_i/n$. Furthermore, Jørgensen et al. [4] only provide solution values for 13 instances: R1a–R3a, R5a, R9a and R10b and R1b, R2b, R5b–R7b, R9b and R10b. Therefore, we also restrict our computations to these instances and we also provide average values over five runs. In order to yield shorter computation times than in our previous experiments, the maximum iteration limit of the VNS was reduced to 25×10^4 iterations.

Table 7 gives the results generated by the proposed VNS. We provide values for the objective function $f'(s)$, total travel distance $c(s)$, total route duration $g(s)$, total passenger waiting time $l(s)$, total excess ride time with respect to direct ride time $r(s)$, the sum over early arrivals $e(s)$ (all other violations are 0 in our case), average ride time, and total vehicle waiting time. In order to re-evaluate the results obtained by the proposed VNS according to $f'(s)$, we determined individual values for $c(s)$, $g(s)$, $l(s)$, $r(s)$, $e(s)$, $w(s)$, $t(s)$, and $d(s)$, where $d(s) = w(s) = t(s) = 0$. These values were then used to compute $f'(s)$ for each VNS solution. We are thus able to directly compare the values provided in the two tables. The according percentage deviations for the objective function values $f'(s)$, provided by Bergvinsdottir [23] and Jørgensen et al. [4], are given in the columns headed with "(%)". A negative percentage deviation indicates that the corresponding average value obtained by means of the VNS is better than the according value computed by the GA. The VNS proposed in this paper yields better results for all instances. Comparing the different components of the objective function at an individual level, the results obtained by the VNS are better in all cases, see Tables 6 and 7. Nevertheless, the large difference in $f'(s)$ is only partly explained by this comparison. It has to be, to a large extent, due to values different from zero for $d(s)$, $w(s)$, and $t(s)$ in case of the GA. Unfortunately, Jørgensen et al. [4] did not report these values. Comparing a weighted combination of the three terms in the objective function where individual values are available for both algorithms ($w_1 c(s) + w_3 l(s) + w_4 g(s)$), the VNS outperforms the GA by, on average, 23%.

In Table 7, we provide average values for total routing costs which correspond to the total distance traveled $c(s)$. When

Table 6
GA by Jørgensen et al. [4] (average values over 5 runs).

	Total cost $f'(s)$	Travel distance $c(s)$	Total duration $g(s)$	Passenger waiting $l(s)$	Average ride time	Vehicle waiting time	CPU[a]
R1a	4696	309	1041	29	19.86	252	5.57
R2a	19426	539	1969	81	28.47	470	11.43
R3a	65306	1047	2779	144	42.79	292	21.58
R5a	213420	1350	4250	286	42.49	500	58.23
R9a	333283	1343	3597	132	57.88	94	40.78
R10a	740890	1811	5006	401	58.42	315	65.98
R1b	4762	284	907	5	26.24	143	5.46
R2b	13580	561	1719	53	25.3	198	11.72
R5b	98111	1344	4296	221	38.46	552	58.93
R6b	185169	1799	5309	361	42.59	630	81.23
R7b	9169	478	1299	27	27.5	102	8.29
R9b	167709	1372	3679	166	49.65	147	44.66
R10b	474758	1740	4733	202	55.34	113	66.41
Avg.	179252.23	1075.15	3121.85	162.15	39.61	292.92	36.94

[a] Run times in minutes on an Intel Celeron 2 GHz processor.

Table 7
VNS (25×10^4 iterations, average values over 5 runs) compared to GA.

	Total cost $f'(s)$	(%)	Travel dist. $c(s)$	Total dur. $g(s)$	Pass. wait. $l(s)$	Excess ride $r(s)$	Early arrival $e(s)$	Avg. ride time	Veh. wait. time	CPU[a]
R1a	**3234.60**	−31.12	273.70	863.65	0.00	49.23	1.40	7.22	109.95	2.70
R2a	**14640.16**	−24.64	431.15	1774.19	0.30	113.52	189.08	7.84	383.04	5.16
R3a	**15969.08**	−75.55	777.92	2425.92	0.42	321.99	88.24	12.54	208.00	6.38
R5a	**23852.00**	−88.82	1023.77	3712.18	0.18	246.75	93.41	8.05	288.42	13.93
R9a	**13806.40**	−95.86	1045.34	3232.18	12.93	529.10	5.66	12.43	26.84	33.53
R10a	**25016.46**	−96.62	1389.73	4499.45	4.02	594.58	52.86	11.35	229.72	40.27
R1b	**2825.53**	−40.66	238.51	737.27	0.00	60.06	0.00	9.51	18.76	3.78
R2b	**5003.11**	−63.16	431.18	1403.74	0.00	25.30	1.54	6.12	12.56	8.29
R5b	**12360.50**	−87.40	951.74	3417.62	0.29	197.15	6.14	7.67	65.88	23.19
R6b	**16499.44**	−91.09	1241.22	4182.99	0.00	275.67	10.83	8.53	61.77	26.39
R7b	**4601.71**	−49.81	359.19	1123.56	0.00	103.33	8.18	9.74	44.37	4.49
R9b	**13412.76**	−92.00	977.59	3211.71	0.02	337.99	12.65	10.36	74.12	30.32
R10b	**16420.00**	−96.54	1310.00	4218.37	1.73	493.34	1.67	10.66	28.38	51.81
Avg.	12 895.52	−71.79	803.93	2677.14	1.53	257.54	36.28	9.39	119.37	19.25

Avg=average, dist=distance, dur=duration, pass=passenger, veh=vehicle, wait=waiting.

[a] Run times in minutes on a Pentium D computer with 3.2 GHz.

compared to the values given in Tables 2 and 5, where only $c(s)$ is minimized, it becomes obvious that a combined objective function, including service quality oriented criteria, such as passenger waiting time or user ride time, entails higher routing costs. This issue is, however, not subject to investigation in this paper. We refer the interested reader to Parragh et al. [12], where routing costs and mean user ride time are treated as separate objectives in a Pareto multi-objective fashion.

5. Conclusion

In this paper we have proposed a variable neighborhood search heuristic for the static multi-vehicle DARP. The basic algorithm is generic enough to be applicable to two different variants of the problem. The only change necessary in the VNS algorithm concerns the evaluation function. In the first version (standard DARP) user inconvenience is represented by time windows and a maximum ride time limit. In the objective function total routing costs, which correspond to the total distance traveled, are minimized. In this problem class the best known results by Cordeau and Laporte [9] are already very good. We were not able to improve them when considering average values over five random runs. However, taking the best out of these five runs, an average improvement of 0.28% was possible. Considering the best solutions across all parameter tuning experiments, 16 new best solutions have been identified.

In the second version, most constraints are relaxed to soft constraints and all aspects of user inconvenience are penalized. This version of the DARP has recently been proposed by Jørgensen et al. [4], who also provided results using a genetic algorithm. In this problem class we achieved remarkable improvements of, on average, 71.79%.

Future research will involve the integration of heterogeneous passengers and vehicles as well as driver related constraints into the problem.

Acknowledgments

This work was supported by the Austrian Science Fund (FWF) under Grant #L286-N04. We thank two anonymous referees for their very valuable comments.

References

[1] Baugh JW, Krishna G, Kakivaya R, Stone JR. Intractability of the dial-a-ride problem and a multiobjective solution using simulated annealing. Eng Optim 1998;30:91–123.
[2] Toth P, Vigo D. Heuristic algorithms for the handicapped persons transportation problem. Transp Sci 1997;31:60–71.
[3] Aldaihani M, Dessouky MM. Hybrid scheduling methods for paratransit operations. Comput Ind Eng 2003;45:75–96.
[4] Jørgensen RM, Larsen J, Bergvinsdottir KB. Solving the dial-a-ride problem using genetic algorithms. J Oper Res Soc 2007;58:1321–31.
[5] Berbeglia G, Cordeau J-F, Gribkovskaia I, Laporte G. Static pickup and delivery problems: a classification scheme and survey. TOP 2007;15:1–31.
[6] Cordeau J-F, Laporte G. The dial-a-ride problem: models and algorithms. Ann Oper Res 2007;153:29–46.
[7] Parragh SN, Doerner KF, Hartl RF. A survey on pickup and delivery problems. Part II: transportation between pickup and delivery locations. J Betriebswirtschaft 2008;58:81–117.
[8] Paquette J, Cordeau J-F, Laporte G. Quality of service in dial-a-ride operations. Comput Ind Eng 2009;56:1721–34.
[9] Cordeau J-F, Laporte G. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. Transport Res B Meth 2003;37:579–94.
[10] Mladenović N, Hansen P. Variable neighborhood search. Comput Oper Res 1997;24:1097–100.
[11] Glover F, Laguna M. Tabu search. Norwell: Kluwer; 1997.
[12] Parragh SN, Doerner KF, Gandibleux X, Hartl RF. A heuristic two-phase solution method for the multi-objective dial-a-ride problem. Networks 2009, to appear, available online, doi: 10.1002/net.20335.
[13] Polacek M, Hartl RF, Doerner KF, Reimann M. A variable neighborhood search for the multi depot vehicle routing problem with time windows. J Heuristics 2004;10:613–27.
[14] Hemmelmayr VC, Doerner KF, Hartl RF. A variable neighborhood search heuristic for periodic routing problems. Eur J Oper Res 2009;195:791–802.
[15] Carrabs F, Cordeau J-F, Laporte G. Variable neighborhood search for the pickup and delivery traveling salesman problem with LIFO loading. INFORMS J Comput 2007;19:618–32.
[16] Cordeau J-F. A branch-and-cut algorithm for the dial-a-ride problem. Oper Res 2006;54:573–86.
[17] Glover F. Ejection chains, reference structures and alternating path methods for traveling salesman problems. Discrete Appl Math 1996;65:223–53.
[18] Ropke S, Pisinger D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transp Sci 2006;40:455–72.
[19] Kirkpatrick S, Gelatt Jr. CD, Vecchi MP. Optimization by simulated annealing. Science 1983;220:671–80.
[20] Savelsbergh MWP. The vehicle routing problem with time windows: minimizing route duration. ORSA J Comput 1992;4:146–54.
[21] Cordeau J-F, Gendreau M, Laporte G. A tabu search heuristic for periodic and multi-depot vehicle routing problems. Networks 1997;30:105–19.
[22] Parragh SN. Ambulance routing problems with rich constraints and multiple objectives. PhD thesis, University of Vienna, Faculty for Business, Economics and Statistics; 2009.
[23] Bergvinsdottir KB. The genetic algorithm for solving the dial-a-ride problem. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark; 2004.