Dynamic programming based metaheuristics for the dial-a-ride problem

Ulrike Ritzinger, Jakob Puchinger, Richard F. Hartl

Annals of Operations Research manuscript No.
(will be inserted by the editor)

Dynamic Programming based Metaheuristics for the Dial-a-Ride Problem

Ulrike Ritzinger · Jakob Puchinger · Richard F. Hartl

Received: date / Accepted: date

Abstract The organization of a specialized transportation system to perform transports for elderly and handicapped people is usually modeled as dial-a-ride problem. Users place transportation requests with specified pickup and delivery locations and times. The requests have to be completed under user inconvenience considerations by a specified fleet of vehicles. In the dial-a-ride problem, the aim is to minimize the total travel times respecting time windows, maximum user ride times, and vehicle restrictions. This paper presents an exact dynamic programming algorithm for the dial-a-ride problem and an heuristic subjected to the dynamic programming concept, which is able to cope with the curse of dimensionality by restricting the considered solution space. Then, a hybrid Large Neighborhood Search is proposed applying the dynamic programming based algorithm. The algorithms are tested on a given set of benchmark instances from the literature.

Keywords Dial-a-Ride Problem · Dynamic Programming · Large Neighborhood Search

Ulrike Ritzinger
Mobility Department, Austrian Institute of Technology
Giefinggasse 2, 1210 Vienna, Austria
E-mail: ulrike.ritzinger.fl@ait.ac.at

Jakob Puchinger
Mobility Department, Austrian Institute of Technology
Giefinggasse 2, 1210 Vienna, Austria
E-mail: jakob.puchinger@ait.ac.at

Richard F. Hartl
Department of Business Administration, University of Vienna
Brünner Straße 72, 1210 Vienna, Austria
E-mail: richard.hartl@univie.ac.at

1 Introduction

The dial-a-ride problem (DARP) generalizes a number of well studied vehicle routing problems such as the Vehicle Routing Problem with Time Windows(VRPTW) in Toth and Vigo (2001) and the Pickup and Delivery Problem (PDP) in Parragh et al (2008). The DARP is a demand responsive transportation service, which arises for example in the context of patient transportation. It is a service for elderly or handicapped people, where patients have to be delivered to medical facilities or carried back home leading to specified pickup and delivery locations. The transportation requests have to be completed under user inconvenience considerations by a given fleet of vehicles. Assigning vehicles of a fleet to given transportation requests is usually modeled in terms of a static DARP, which has obtained noticeable attention in the literature in recent years. In contrast to other vehicle routing problems such as the PDP or VRPTW, in the DARP humans have to be transported instead of goods. Therefore, the aim besides minimizing the operating costs is to ensure patient convenience. This is done by introducing additional constraints such as the maximum user ride time or to specify tight time windows. There exist many variants of the DARP depending on the specific application. An excellent overview about considered problem variants and existing solution methods of this problem class can be found in Cordeau and Laporte (2007) and in Parragh et al (2010a).

In the classic DARP the objective function corresponds to minimizing the total routing costs. Cordeau and Laporte (2003) proposed a Tabu Search algorithm to solve this variant of the problem. Ropke et al (2007) present a Branch-and-Cut algorithm for solving the Pickup and Delivery Problems with time windows, as well as, exactly solving the easier instances of the benchmark data set for the DARP. Parragh et al (2010b) report improved results for related benchmark instances obtained by means of a Variable Neighborhood Search. Parragh and Schmid (2013) present a hybrid approach combining Column Generation, Large Neighborhood Search, and Variable Neighborhood Search, improving currently best known solutions.

There is a growing interest in fast methods for obtaining high quality DARP solutions, since DARPs often occur in a dynamic real-world setting. An example for such a dynamic setting can be found in Schilde et al (2011) where it is investigate whether the use of stochastic information about future requests can improve solution quality.

Dynamic Programming (DP) is a well known exact method for solving complex problems by decomposing the problem into a number of smaller subproblems. The first DP algorithm for a single vehicle, many-to-many, immediate-request DARP is given by Psaraftis (1980). Gromicho et al (2012) present a flexible framework for solving realistic VRPs. The solution approach within this framework is a generalization of the restricted DP heuristic for the Traveling Salesman Problem of Malandraki and Dial (1996).

Most of the successful algorithms for solving complex transport optimization problems such as the dial-a-ride problem rely on a combination of several

methodological approaches for solving large problem instances. An overview as well as success stories of hybrid metaheuristics can be found in Blum et al (2011) and Talbi (2013). They are often used to solve complex and large real-world optimization problems, combining advantages from various fields of computer science and mathematical optimization. In the recent vehicle routing literature there is an increasing number of successful applications of hybrid metaheuristics Gendreau et al (2008). The specific area of applying hybrid metaheuristics to dynamic and stochastic vehicle routing problems has been examined in Ritzinger and Puchinger (2013).

According to these developments, we present a DP algorithm for the multiple vehicle DARP which is able to solve small instances of the benchmark data set to optimality. Due to memory constraints we adapt this algorithm to a restricted dynamic programming heuristic, able to solve instances with more requests in short computational time. As shown in the literature, Large Neighborhood Search (LNS) is a promising concept to solve routing problems in general (Pisinger and Ropke (2010)), pickup and delivery problem with time windows (Ropke and Pisinger (2006), Bent and Hentenryck (2006)) and dial-a-ride problems (Parragh and Schmid (2013)), we propose a hybrid heuristic which integrates the dynamic programming based algorithms into a LNS framework. The main contribution of this article is the development and analysis of dynamic programming based algorithms for solving dial-a-ride problems.

The remainder of this article is organized as follows. Section 2 gives a formal description of the DARP. Section 3 is devoted to a detailed description of the proposed algorithms and the LNS solution framework. This is followed by the last section presenting computational results for the proposed methods. Finally, conclusions and an outlook towards future work is given.

2 The Dial-a-Ride Problem

In this work, we consider the DARP as defined in Cordeau and Laporte (2003). This formulation considers a given homogeneous fleet (all vehicles are of the same type), as well as, homogeneous transportation requests. The notations and formulations introduced in this section are used throughout the paper.

The static DARP is modeled as a complete graph $G = (V, A)$, where $V = \{v_1, \ldots, v_{2n}\}$ is the set of all nodes and $A = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$ is the set of all arcs. Each arc $a(v_i, v_j)$ is associated with a non-negative cost value representing the static travel time t_{ij} from node v_i to v_j. The set of nodes covers n patient requests, where each request consists of a pair of pickup and delivery location (v_i, v_{n+i}). The given patient requests have to be served by m vehicles with a maximum load Q and a maximum tour duration T. All the vehicles are based at a single depot, introducing node v_0 for the origin depot and v_{2n+1} for the end depot. Each node v_i is associated with a load q_i, defining the number of patients to be loaded or unloaded at this node. It is not allowed that q_i exceeds Q at any time.

For each patient request a time window $[e_i, l_i]$ is set, determining at what time the service must take place, where e_i defines the earliest begin of service time and l_i is the latest possible begin of service time. The time window $[e_i, l_i]$ is either given at the pickup location or the delivery location, depending if it is an *outbound* or *inbound* request. An *outbound* request brings the patient from its home to the hospital. Because here it is important that the patient arrives at the right time in the hospital, the time window is given at the delivery location. In case of an *inbound* request, where the patient gets a ride back home, the time window is given at the pickup location to avoid long waiting times for the patients. Respectively, the time window for the other location is set to $e_i = 0$ and $l_i = T$. The service time for loading or unloading a patient is denoted by d_i. In order to ensure patient convenience a maximum user ride time L is set, defining the time a patient is allowed to spend on the vehicle at most. The user ride time starts after loading the patient into the vehicle, thus, their own service times are not included. By introducing the maximum user ride time constraint long detours for the patients can be avoided. For the depot a time window $[e_0, l_0]$ is set which specifies the time when vehicles are allowed to service transportation requests. The duration of a route r results from the time between the vehicle leaves the depot and returns back to it and must not exceed the given maximum route duration T.

The arrival time of a vehicle at node v_i is denoted as A_i, the begin of service time by B_i and the departure time by $D_i = B_i + d_i$. In the case a vehicle arrives to early at node v_i it has to wait until the time window starts: $B_i \geq max\{e_i, A_i\}$. Waiting $W_i = B_i - A_i$ at a vertex v_i is only allowed before the service starts but not after the service has finished. The departure at a node takes place after loading the patient into the vehicle with departure time $D_i = B_i + d_i$ and the load on the vehicle Q_i. In the case $B_i > l_i$ the time window constraint is violated. The user ride time associated with request i is $L_i = B_{n+i} - D_i$, thus, the time between the end of service at node v_i and the beginning of service at v_{n+i} must not exceed L. Without the user ride time constraint it would be always optimal to set $B_i = max\{e_i, A_i\}$. However, in the case a patient is aboard the aim is to reduce unnecessary waiting times, thus, it could be better to start the service at a later point in time. A graphical summary of the relevant times in the DARP problem can be found in Figure 1.

The aim in the DARP is to construct vehicle routes such that all patient requests are served in time and by respecting all the given constraints. The objective function is to minimize the total travel times of all vehicles: $\sum_{a(v_i,v_j)\in S} t_{ij}$, with $a(v_i, v_j)$ depicting the used arcs in the solution S.

3 Solution Framework

The proposed hybrid solution framework for solving the DARP includes two algorithmic components: dynamic programming (DP) and Large Neighborhood Search (LNS). First, the exact DP algorithm and the restricted DP heuristic are described. These two algorithms are the main building blocks of a hy-

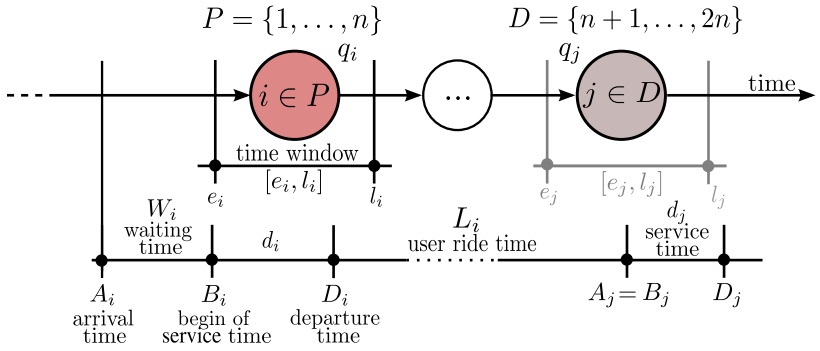

Fig. 1 Relevant time information in the Dial-a-Ride Problem

brid solution framework incorporated into a LNS. The combination is then illustrated in further detail.

3.1 Dynamic Programming Algorithm

The proposed exact DP algorithm for the DARP is based on the exact DP algorithm for the Traveling Salesman Problem (TSP) introduced by Held and Karp (1961) and Bellman (1962). There, the problem is to visit a set $V = \{v_0, \ldots, v_n\}$ of cities exactly once where the distances between each pair of cities $(v_i, v_j), v_i, v_j \in V, v_i \neq v_j$ are given by t_{ij}. The tour starts and ends in city v_0 and the objective is to minimize the total distance traveled. In dynamic programming *states* are defined to represent the current status of a problem. For the TSP, a state $(S, v_j), v_j \in S, S \subseteq V \setminus v_0$ defines a path starting in node v_0, visiting all nodes in S exactly once and ending in node v_j. The costs of such a state are defined by $C(S, v_j)$, holding the minimum travel distance of such paths. In the first stage the costs of the states are computed by $C(\{v_j\}, v_j) = t_{0j}, \forall v_j \in V \setminus v_0$. In the next stages the costs for each state are calculated by recursion: $C(S, v_j) = min_{v_i \in S \setminus v_j}\{C(S \setminus v_j, v_i) + t_{ij}\}$. The optimal TSP tour, visiting all cities and ending in the depot is given by $min_{v_j \in V \setminus v_0}\{C(V \setminus v_0, v_j) + t_{j0}\}$.

To represent the current status in the DARP the given formulation has to be extended as follows. For the DARP, a state $\mathfrak{S}_j = (S, v_j, B_j, U_j)$ encodes a path consisting of a set of already visited nodes S and a last visited node v_j. One extension of the state information is given by B_j which stores the earliest begin of service time at the last visited node v_j. This time is needed to respect the time window constraints and must be $B_j \leq l_j$ for any node $v_j \in S$. Additionally, information about the user ride times of the patients aboard has to be tracked. U_j is a set of the effective user ride times for all patients aboard arriving in node v_j. For each patient loaded at the vehicle $L_j \leq L, L_j \in U_j$ must be applied for any feasible state. The cost of a state $C(S, v_j, B_j, U_j)$ is also defined by the travel distance of the path it is representing.

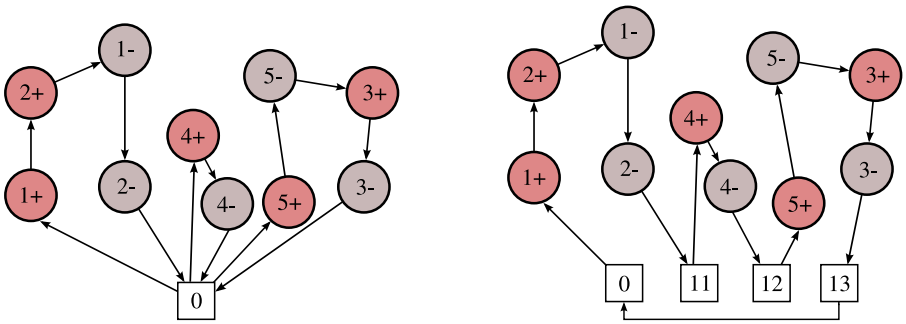

Fig. 2 An example tour of three vehicles $m = 3$ and five transportation requests $n = 5$, where the pickup nodes are depicted as $1^+, 2^+, \ldots, 5^+$ and the delivery nodes as $1^-, 2^-, \ldots, 5^-$, is shown in the left picture. The corresponding simplified giant tour representation introducing an additional node for each vehicle is illustrated on the right.

Other than in the TSP, the DARP has to deal with more than one vehicle. To handle multiple vehicles the idea of a giant tour representation (GTR) is proposed by Funke et al (2005). In the GTR, for each vehicle $w = 1, \ldots, m$ an origin node o_w and a destination anode d_w is introduced. This allows a routing solution in the graph where each route end node d_w is connected to the route start node of the next vehicle o_{w+1}. Finally, the route is closed by connecting d_m with o_1. In order to be able to solve the multiple vehicle DARP a slight adaptation of the GTR is done. While the GTR described in Funke et al (2005), is applicable to VRPs with multiple depots and/or problems considering a heterogeneous fleet, we have a problem where only a single depot and a homogeneous fleet is considered. This allows us to simplify the given GTR by adding only one additional node for each vehicle, resulting in a set $M = \{v_{2n+1}, \ldots, v_{2n+m}\}$ which is added to V. In that case, a node $v_j \in M$ represents as many depot nodes as vehicles are given. An example for the used GTR is depicted in Figure 2. In the state representation one can say it shows the change from one vehicle to the next one by setting $S = \emptyset$, $U_j = \emptyset$, $v_j = v_0$ and $B_j = e_0$.

The algorithm follows the concept described above and starts with node v_0 which represents the first vehicle at the depot and determines the costs of all states $C(\{v_j\}, v_j, B_j, U_j) = t_{0j},\ \forall v_j \in V$, which are all not yet visited nodes. In each successive stage, all states of the current stage are expanded with all not yet attended nodes by recursion $C(S, v_j, B_j, U_j) = min_{v_i \in S \setminus v_j}\{C(S \setminus v_j, v_i, B_i, U_i) + t_{ij}\}$. The expansion of states is done as long as all nodes are visited once. To finish the tour correctly, the last expansion is the way from the last visited node back to the depot. The optimal solution is given by $min_{v_j \in V \setminus v_0}\{C(V \setminus v_0, v_j, B_j, U_j) + t_{j0}\}$.

To ensure the feasibility of the solution, several constraints need to be considered at the expansion of the states. First of all, the precedence of pickup and delivery nodes must be considered, thus, node v_i must always be visited before node v_{n+i}. In the DP algorithm this means that each state $\mathfrak{S}_j$ is expanded by

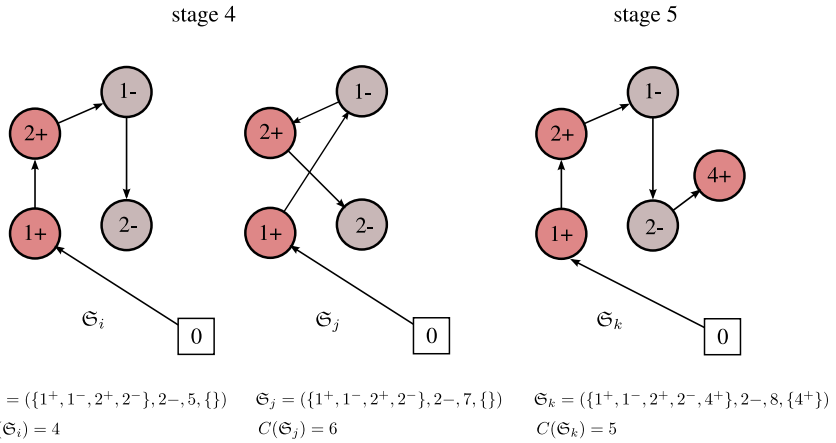

Fig. 3 States in the DP algorithm, where $\mathfrak{S}_j$ is dominated by state $\mathfrak{S}_i$. Because they have the same set of visited nodes and the same last visited node the dominance check can be applied. Here, $\mathfrak{S}_i$ has smaller costs and an earlier begin of service time in node 2^-. Thus, $\mathfrak{S}_j$ can be discarded and $\mathfrak{S}_i$ is considered in the next stage 5.

all pickup nodes $v_1, \ldots, v_n$ first and all delivery nodes $v_{n+i} \in \{v_{n+1}, \ldots, v_{2n}\}$ where the corresponding pickup node v_i has been already visited. Additionally, all the time window constraints must be respected $B_i \leq l_i$, as well as, the maximum route duration $B_{2n+m} - D_0 \leq T$ and the maximum load Q of a vehicle. One of the most challenging computational aspects of the DARP are the user ride time constraints $L_i \leq L$. This is caused by the fact that postponing the start of a service can render a path feasible with respect to the user ride time constraint. In the forward recursion of the DP algorithm, the user ride time check can be applied by computing the ride time u_{ij} from the actual node v_i to the next expanded node v_j, where the ride time u_{ij} is defined as $u_{ij} = max\{t_{ij}, B_j - (l_i + d_i)\}$ and add it to the current user ride times of the loaded patients $L_i + u_{ij}$. This gives us a lower bound for the user ride time whereas the upper bound is given by tightening the time windows as described in the next paragraph.

While executing this DP algorithm, a huge number of states are opened (curse of dimensionality). It is possible to reduce the number of considered states by introducing several tightening constraints, loosely based on the inequalities in Ropke et al (2007) and a dominance rule as in Dumas et al (1995).

Tighten time windows. The first preprocessing step is to tighten the time windows of all nodes where only a loose time window is set, according to the corresponding node. In the case of an *outbound* request, where the time window for the delivery node is given, the time window for the pickup can be set as follows: $\bar{e}_i = e_j - L - s_i$ and $\bar{l}_i = l_j - t_{ij} - s_i$. In the other case, given the time window for the pickup location (*inbound*), the time window at the delivery location is set to $\bar{e}_j = e_i + s_i + t_{ij}$ and $\bar{l}_j = e_i + L + s_i$.

User ride time. The next check is considering user ride time. Whenever patients are loaded on a vehicle we examine, before expanding the state by the next node v_i, whether a path can be found to deliver all patients aboard without violating their user ride times. If there exists at least one feasible path the expansion by v_i will be done, otherwise, this state is not opened (even in the case, the expansion of v_i itself would be feasible). At the point where the last vehicle is on tour, a check about the reachability of prospective nodes can be done. Thus, whenever the expansion by a node makes it impossible to reach all other non-visited nodes, it will not be executed because no feasible solution can be reached.

Dominance rule. Applying dominance rules to the algorithm further reduces the number of considered states. Whenever, it happens that two states have visited the same node set $S_i \cap S_j = \emptyset$ and end up in the same last visited node $v_i = v_j$, the possibility is given to discard the worse one, and consider only the better state for further expansion steps. In Dumas et al (1995), dominance rules are introduced guaranteeing the optimality of the solution. In our case state $\mathfrak{S}_i$ dominates another state $\mathfrak{S}_j$, if the costs are less $C(\mathfrak{S}_i) < C(\mathfrak{S}_j)$, the begin of service time is earlier $B_i <= B_j$ and the user ride time of each patient aboard is smaller $L_i <= L_j, \forall i \in U_i, j \in U_j$. Only, if all these constraints are fulfilled it is allowed to discard the worse state, otherwise both states have to be expanded in the next stage. In Figure 3 an example is given at that.

Simple GTR. Another way of reducing the number of states is the simplification of the GTR. Thus, only one expansion has to be done for ending a route in the depot node and starting a new one in the same node. Adding these tightening constraints and the dominance rule to our DP algorithm allows us to solve small instances of the benchmark data set exactly.

3.2 Restricted DP Algorithm

In order to tackle larger instances, the exact DP algorithm is adapted to a restricted DP heuristic. In Malandraki and Dial (1996) a restricted DP algorithm for the TSP is proposed. The main concept is to bound the number of considered states in every stage by a given parameter B. Since each state represents a path and all paths in a stage have visited the same number of nodes, it is more likely that a state with lower costs will result in a good solution than one with higher costs. Thus, the heuristic selects B states in each stage with the lowest costs and considers only these B states for further expansion in the next stage. Note that taking $B = 1$ results in a nearest neighborhood heuristic, while setting $B = \infty$ makes an exact DP algorithm. Gromicho et al (2012) propose a restricted DP algorithm for solving realistic VRPs and restrict the state space even further, by applying a form of beam search (Norvig (1992)), which means that a state is only expanded by its E nearest feasible

nodes. It has to be noticed that making the state selection too greedy might prevent the algorithm from finding a feasible solution at all.

To restrict the state space for the DARP we examined various methods for selecting the best B states of a stage. Because of the time window and user ride time constraints it is not sufficient to select the states with the smallest costs. Hence, we introduce a selection function $s(\mathfrak{S}_i)$ for each state taking global information into account. It estimates how promising a further expansion of that state might be. The function $s(\mathfrak{S}_i)$ comprises the actual costs of a state, a simple estimation about the required travel distance to serve all remaining nodes and the average remaining time per open request. This results in:

$$s(\mathfrak{S}_i) = C(\mathfrak{S}_i) + \sum_{(i,j)\in V\setminus S} t_{0i} + t_{ij} + t_{j0} + \sum_{m}(T_m - A_j^m)/|V \setminus S| \quad (1)$$

We also tested various other selection functions, as for example, only considering the travel distance, which turns out to be insufficient for most of the instances. Another approach is to incorporate the accumulated waiting times of the vehicles. A comparison of these various functions is given in Section 4. We also tried to improve the estimated travel time of the not yet visited nodes, for example, by applying a nearest neighborhood heuristic, according to time and distance, which ignores only the user ride time constraint, as well as, a minimum spanning tree algorithm. But it turns out that this functions are to time consuming and in addition, did not improve the solution quality.

Finally, the best B states according to function (1) are selected and expanded in each stage. In addition, the beam search concept is added to the restricted DP heuristic by selecting the nearest nodes in time. However, for the DARP beam search only has an effect when the vehicle is empty and all possible pickup locations can be expanded. Otherwise the expansion is limited to the time windows and user ride times determined by the passengers aboard. Even though the optimality guarantee is no longer given when running the restricted DP heuristic, we are able to find good solutions for a given benchmark data set and even the optima for the small instances in short computational time.

3.3 Large Neighborhood Search

A solution of the DARP consists of a list of tours, one for each vehicle. In each tour a subset of transport requests is completed by picking up a patient at its location within the given time window and drop the patient off at his desired location without exceeding the maximum user ride time. Such a feasible initial solution can be obtained by the restricted DP algorithm as discussed in the previous section. One disadvantage of the restricted DP algorithm is its sequential structure, which means that one vehicle is considered after another. Once a vehicle returns to the depot there is no possibility to insert a new request in a later point in time. Even tough, a selection function which takes

global information into account is applied, the algorithm is challenged by taking a decision at an early point in execution time. So for example, considering large instances, the algorithm has to apply the selection function already in the third stage, but in this stage there are only two nodes visited, and according to that information the selection has to be done. However, the big advantage lies in gaining a feasible solution for the DARP in short computational time. To further improve these results, a hybrid metaheuristic is introduced which combines the above described DP algorithm and a LNS.

The LNS is introduced by Shaw (1998) and follows a simple principle. The idea is, to partially destroy and repair the incumbent solution in each iteration. The destroy operator removes a given number of requests and the repair operator reinserts those requests. Every time an improved solution is found, it is used in the next iteration. An extension of this approach is proposed in Ropke and Pisinger (2006), where various destroy and repair operators are used for the VRP with time windows. Parragh and Schmid (2013) present a successful LNS for the DARP, where the operators correspond to the ones from Ropke and Pisinger (2006). Some of these operators are be incorporated into our framework. Additionally, we introduce new operators based on the previously presented DP algorithms. This results in three different operator types described in the following sections. Ropke and Pisinger (2006) propose an adaptive layer to guide the selection of the operators, whereas Parragh and Schmid (2013) select them at random. In our framework we also used a random selection strategy, first, for selecting the type of the operator to be applied and second, for the operators itself.

3.3.1 Request based heuristics

Every time this operator type is selected a number of requests r has to be removed from the incumbent solution s'. The number r is determined at in every such iteration by randomly selecting a value between $10 - 30\%$ of the number of nodes. For all r requests, the removal operator removes the pickup and delivery locations and stores it into a list of yet unassigned requests (request bank).

- **Random.** The random removal operator randomly removes r requests from the incumbent solution, by randomly selecting a tour and a pickup location.
- **Worst.** The worst removal operator iteratively removes the request whose removal improves the costs most. Thus, in each iteration r the worst request is determined and removed from the solution.
- **Related.** Finally, the related removal operator removes related requests from the solution. The measure for similarity is nearly the same as in Parragh and Schmid (2013), also considering temporary and geographical closeness, but taking e_i the given earliest begin of service time instead of B_i, the actual begin of service time in node v_i. Two requests i and j are related if $(|e_i - e_j| + |e_{n+i} - e_{n+j}| + t_{ij} + t_{n+i,n+j})$ is small. The first removed

request i is removed randomly. Then, the $r-1$ most related requests to i are removed and stored to the request bank as well.

All requests stored in the request bank are reinserted iteratively by repair operators in the next step. It is randomly determined whether a *greedy*, a *2-regret* or a *3-regret* insertion heuristic is used, with a higher probability for the two *regret* insertion heuristics. Here, every time a request from the bank has to be inserted, the exact DP algorithm is used. For each vehicle a DP algorithm is run with its currently assigned requests and the removed one as input. The request is then inserted into the tour, where the costs are deteriorated at least. Since the DP algorithm has to solve an instance with a single vehicle only, it is very fast and the obtained insertion is optimal for the considered vehicle.

- **Greedy.** In the greedy insertion heuristic the unassigned request whose insertion causes the least cost increase is inserted at its best possible position.
- **Regret.** In the regret insertion, the request which has the largest regret value is inserted at its best possible position. The *2-regret* value represents the cost difference of inserting the request i at its best position or second-best position. The *3-regret* value is defined analogously. cost difference of inserting the request i at its best position or third-best position. More details about this heuristic are given in Ropke and Pisinger (2006).

3.3.2 Block based heuristics

The idea of the block based operator type is to reapply the DP algorithm on the incumbent solution. The initial idea was to remove requests from the solution using one of the removal operators and restart the DP algorithm keeping the remaining tours fixed. The problem is, that with the forward recursion, it is ensured that none of the constraints are violated, but no exact arrival or departure times are stored, thus, new requests can not be easily inserted.

As shown in Parragh et al (2010b), the zero split neighborhood is a powerful tool for solving the DARP. A zero split is a special sequence of requests in a tour, which lies between two arcs where the vehicle load is zero, thus, no patient is aboard. Every route has at least one such sequence, but the authors discovered that most of the time there is more than one sequence of this type in a route. The big advantage of the zero splits is, that such a sequence does not violate the user ride time constraint in our intermediate solutions, since we are starting with a feasible solution from the restricted DP heuristic.

The basic concept of the block based heuristic is to destroy the incumbent solution into all its zero split sequences (alias blocks) and create for each such sequence a new *block-node*. The vehicle routing problem induced by those *block-nodes* is then solved using the DP algorithm. This is a very powerful operator because the whole solution might be destroyed and repaired. Before we describe the different operators in more detail, we illustrate the variables required to be computed for a new *block-node*. For each block $\mathfrak{B}$, the path of already visited nodes $B = \{v_1, v_2, v_{n+1}, \ldots, v_{n+2}\}, B \subseteq S$ and the earliest

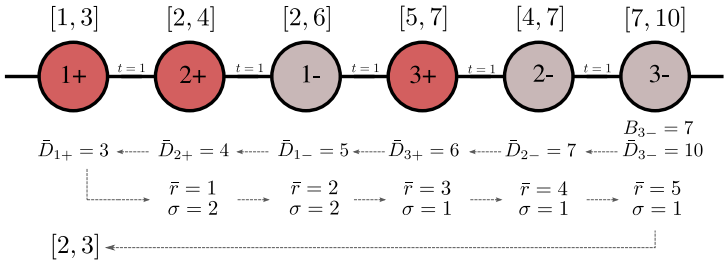

Fig. 4 Calculation of the time window for the first node $v_{1+} = [2, 3]$ of a block B.

begin of service time at the last node B_{n+3} are known from the DP solution. The question now is, in what time window $[\hat{e}_{v_1}, \hat{l}_{v_1}]$ must the service take place at the first node v_1 to maintain a feasible zero split sequence. The according time window $[\hat{e}_{v_1}, \hat{l}_{v_1}]$ can be obtained as follows. First, the latest possible departure time $\bar{D}_{v_1}$ for the first node in the block is computed by backward calculation from the last node via $\bar{D}_{i-} = min\{l_i, (\bar{D}_i - t_{i-,i} - d_i)\}$. Given $\bar{D}_{v_1}$, the minimum duration $\bar{r}$ of the block can be determined by a forward calculation: $\bar{r}_{i+1} = max\{e_{i+1}, B_{i+1}\} - \bar{D}_{v_i}$ with $B_{i+1} = d_i + t_{i,i+1}$. During the forward calculation we also store a value $\sigma(\bar{r}) = min(B_i - e_i), \forall i \in B$. With it, the time window of the first node v_1 in the block can be set to $[\hat{e}_{v_1}, \hat{l}_{v_1}]$, where $\hat{e}_{v_1} = \bar{D}_{v_1} - max\{0, \sigma(\bar{r})\}$ and $\hat{l}_{v_1} = \bar{D}_{v_1}$. An example is given in Figure 4. There, the earliest begin of service time at the last node $B_{3-} = 7$ resulting from the DP algorithm. Then, the latest possible departure times in each node are calculated and yield to $\bar{D}_{1+} = 3$. Starting within this time window has an effect at the arrival time at the last node in the block, whereas starting before this time window results in only increasing the duration of the block but not affecting the arrival time at the last node. Setting this time window in the *block-node* and observing it in the DP algorithm ensures that no constraint of the DARP for this the sequence will be violated.

For the LNS the following operators which are based on the zero split principle are introduced:

- **Single-block.** The first destroy operator is the single-block operator. It destroys the incumbent solution into all its zero split sequences and creates a *block-node* for every sequence.
- **Break-block.** The other destroy operators are called break-block operators. Here, the solution is also destroyed into its blocks, but additionally one or more blocks are selected and decomposed into their single nodes. In a first step, a randomly selected number $q = [1, \ldots, 6]$ determines how many blocks should be decomposed. Then, the blocks are either chosen randomly or by their relatedness of the according first nodes in the blocks.
- **Block-repair.** The block-repair operator takes a set of *block-nodes* and runs a DP algorithm to reassign the blocks. If the break-block operator has been used for destruction, a DARP consisting of *block-nodes* and original nodes is solved. The problem is solved exactly as if the size of the problem

is smaller than 25 blocks and/or requests. If the porblem is larger, it is solved by applying the restricted DP. In that case, the selection function is modified to $\bar{s}(\mathfrak{S}_i) = C(\mathfrak{S}_i)/|S| + \sum_{(i,j)\in W}(t_{0i} + t_{ij} + t_a j0)/|W|$, with $W = V \setminus S$, because here, the states in a stage do not necessarily contain the same number of visited nodes.

3.3.3 Tour based heuristics

The last operator type rearranges multiple requests of several routes at once. It removes a given number of tours from the incumbent solution and runs the restricted DP heuristic on the arising subproblem. We define four different operators, from which one is chosen randomly. The *2-best* operator selects two routes of the solution and rearranges the according requests via the restricted DP heuristic. This is done for all combinations of routes and the one with the best improvement is returned. For larger instances this can be very time consuming, we therefore define the *2-random*, *3-random* and *4-random* operators, where two, three or four tours are removed from the solution at random and a restricted DP heuristic with these requests and the according number of vehicles is applied.

3.3.4 Local search

After applying one of the described destroy and repair operators, the new solution undergoes a local search improvement step if its costs is at most 5% worse then the current best solution. Only with a chance of 1% a local search step is done for new solutions which are worse than the given threshold.

The local search operator tries to shift a sequence of requests with length $q = 1, \ldots, 4$ to other tours. The length n of the sequence is chosen randomly. Then, every possible sequence is removed from its tour, and the removed nodes are reinserted into all the other tours one by one via the DP algorithm at its best position. For small instances this is applied for all tours, while for larger instances it is applied to half of the tours which are randomly chosen. Additionally, there is a time limit of 1 minute set for the local search.

4 Computational Results

All of the developed algorithms have been implemented in Java 7 and the tests were completed on a Intel Xeon 2.4 GHz with 8GB of RAM. The algorithms were tested on benchmark data sets introduced by Cordeau and Laporte (2003) and Ropke et al (2007), which contain randomly generated Euclidean DARP instances. There are three different data sets: a-instances, b-instances and pr-instances. The a-data set includes instances with up to 8 vehicles and up to 96 requests. At each node, a patient can be loaded or unloaded with a service time of 1 and the maximum user ride time is 30 minutes. The maximum capacity of the vehicles is limited to 3 and the time windows are tight with a range of 15

minutes. In the b-data set the maximum capacity is up to 6 and the maximum user ride time is 45 minutes. The time windows are tight and the load at each node can be up to 6. For solving these instances, a branch-and-cut algorithm has been introduced in Ropke et al (2007), obtaining optimal solutions for all instance, whereas in the earlier paper of Cordeau and Laporte (2003) a Tabu Search (TS) algorithm has been proposed. The third set (pr-instances), contains instances which are even more challenging to solve with a fleet size of up to 13 and up to 144 requests. The maximum load of a vehicle is 6 and the maximum route duration is limited to 480. At each node a single patient can be loaded or unloaded and the maximum user ride time is set to 90 minutes. In the first 10 instances, narrow time windows are considered, whereas, for the second 10 instances wider time windows are given. The currently best known approaches for solving these instance area a Variable Neighborhood Search (VNS) proposed in Parragh et al (2010b) and a hybrid Large Neighborhood Search introduced in Parragh and Schmid (2013).

4.1 Dynamic Programming Approaches

The exact DP algorithm, with all the tightening constraints applied, manages to solve instances with up to 24 requests. Due to memory restrictions, we have to use the restricted DP algorithm with an appropriate parameter B to obtain feasible solutions for all instances, and even optimal solutions can be found for some of the instances. The detailed results for all three data sets of the restricted DP algorithm are summarized in the according tables in the column "Restricted DP" in Table 2 for the a-instances, in Table 3 for the b-instances and in Table 4 for the pr-instances. These tables show in the first block (Literature) the results of the hybrid LNS approach proposed in Parragh and Schmid (2013), in the second block (Restricted DP) the results of our restricted DP algorithm and in the third block (Hybrid LNS) the results of our hybrid LNS algorithm. Respectively, each block depicts the best found solution (Best), the average solution (Avg.) and the runtime (CPU) in minutes. The average solution value of the results of the restricted DP algorithm and our proposed hybrid LNS heuristic are compared to the results (averages) of the hybrid LNS in Parragh and Schmid (2013).

It is shown that the restricted DP algorithm produces results which are overall 9.5% worse for the a-instances, 8.5% worse for the b-instances and 16.8% worse for the pr-instances. The advantage of the restricted DP algorithm lies in its short computational time. For the a-instances, the average runtime for the restricted DP algorithm is 0.73 minutes, whereas the LNS heuristic from the literature has an average runtime of 2.26 minutes. For the b-instances it is 0.78 minutes to 2.25 minutes and for the third set the average runtime is 2.24 minutes to 21.84 minutes. Compared to the results of the branch-and-cut algorithm from Ropke et al (2007), we are able to compute good quality solutions requiring much less running time. However, the solution quality is about 10% worse than the optimal solutions.

Instance Set	#solutions found				solution quality			
	s_0	s_1	s_2	s_3	s_0	s_1	s_2	s_3
CL03	6	18	20	19	94.5%	99.9%	92.5%	92.9%
Cor06	11	23	24	24	96.7%	100.0%	92.2%	91.3%
Cor06+	6	20	24	23	98.8%	100.0%	94.2%	93.4%
total	23	61	68	66	96.6%	100.0%	93.0%	92.5%

Table 1 Results for the selection function used in the restricted DP algorithm

In the following we analyze the effects of using various selection functions in the restricted DP algorithm. In Table 1 the results obtained by four different selection functions are compared. The first selection function $s_0(\mathfrak{S})$, selects the states according to their smallest travel distance. The second function $s_1(\mathfrak{S})$ is the defined in (1), considering travel distance, estimated travel distance of open nodes and an estimation of remaining time per open requests. The other two functions $s_2(\mathfrak{S})$ and $s_3(\mathfrak{S})$ consider in addition to the travel distance the accumulated waiting times of the vehicles. The difference is, that in s_2 the estimated travel distance of the not yet visited nodes is incorporated, while in s_3 not. The table shows, how many times a feasible solution is found with the according selection function by setting $B = 10.000$. We chose this value, because this setting allows us to find a feasible solution for all instances in short computational time. The first block (#solutions found) shows, that using selection function s_2 allows the restricted DP algorithm to find feasible solutions for all benchmark instances, whereas the results with s_0 are rather poor (23 times out of 68). The second block (solution quality) shows that by applying the selection function s_1, the best solution qualities can be obtained. Note that the solution quality of s_0 seems to be high, this is due to the fact that only feasible solutions are taken into account. According to these results, the default setting for the restricted DP algorithm are set to $B = 10.000$ and the default selection function s_1 is chosen. It the case, no feasible solution could be found, the algorithm will be executed again with selection function s_2.

4.2 Large Neighborhood Search

In the LNS framework a feasible starting solution is obtained from the restricted DP algorithm. The LNS is run 10 times for each instance with 100 iterations, where in each iteration one of the destroy and repair operators (request-based, block-based, tour-based) is chosen randomly with equal probability. After applying one of the described destroy and repair operators, the new solution undergoes a local search improvement step if its costs is at most 5% worse then the current solution. Only with a chance of 1% a local search step is performed for solutions which are worse than the given threshold. In

			Literature			Restricted DP				Hybrid LNS			
Instance	m	n	Best	Avg.	CPU	Best	Avg.	CPU	%	Best	Avg.	CPU	%
a2-16	2	16	294.25	294.25	0.12	294.25	294.25	0.02	-0.0	294.25	294.25	0.05	0.0
a2-20	2	20	344.83	344.83	0.28	344.83	349.64	0.04	1.4	344.83	344.83	0.11	0.0
a2-24	2	24	431.12	431.12	0.35	433.56	449.19	0.05	4.2	431.12	431.12	0.11	0.0
a3-24	3	24	344.83	344.83	0.29	344.83	366.08	0.10	6.2	344.83	344.83	0.95	0.0
a3-30	3	30	494.85	495.27	0.50	495.56	509.82	0.12	2.9	494.85	494.85	0.90	-0.1
a3-36	3	36	583.19	583.25	0.83	628.31	641.99	0.27	10.1	591.09	595.94	0.39	2.2
a4-32	4	32	485.50	485.71	0.55	493.21	504.71	0.17	3.9	486.57	486.57	1.97	0.2
a4-40	4	40	557.69	557.69	0.78	584.32	633.00	0.31	13.5	557.94	558.98	0.75	0.2
a4-48	4	48	668.82	669.04	1.62	679.43	736.56	0.61	10.1	672.03	673.67	1.03	0.7
a5-40	5	40	498.41	498.41	0.85	504.31	547.23	0.28	9.8	498.41	498.41	8.76	0.0
a5-50	5	50	686.63	686.87	1.60	705.05	736.28	0.56	7.2	687.57	691.14	1.30	0.6
a5-60	5	60	808.48	809.34	2.51	821.74	897.80	0.91	10.9	808.84	814.53	1.90	0.6
a6-48	6	48	604.12	604.54	1.14	614.74	666.59	0.40	10.3	606.26	609.49	1.16	0.8
a6-60	6	60	820.30	821.91	2.29	846.50	927.65	0.88	12.9	821.88	825.00	1.74	0.4
a6-72	6	72	916.66	919.77	4.43	957.21	1047.89	1.41	13.9	925.75	929.78	3.23	1.1
a7-56	7	56	724.04	724.04	1.67	762.44	800.82	0.78	10.6	728.30	732.94	1.83	1.2
a7-70	7	70	889.58	893.50	2.88	947.80	1026.50	1.36	14.9	895.07	904.15	3.07	1.2
a7-84	7	84	1036.17	1040.39	7.04	1091.85	1182.52	1.74	13.7	1043.83	1055.38	4.63	1.4
a8-64	8	64	747.46	747.99	2.14	789.81	843.73	1.07	12.8	755.13	759.94	1.41	1.6
a8-80	8	80	948.69	951.36	5.73	1016.73	1087.30	1.67	14.3	957.08	964.82	2.10	1.4
a8-96	8	96	1234.78	1236.27	9.92	1308.54	1434.44	2.52	16.0	1252.82	1260.80	3.16	2.0
avg			672.40	673.35	2.26	698.34	746.86	0.73	9.5	676.12	679.59	1.93	0.7

Table 2 Results for the hybrid LNS and restricted DP algorithm for a-instances

order to diversify the search, solutions which impair the current solution by at most 3% are accepted with a probability of 10%. Since worse solutions are allowed to be accepted, the best solution found so far is stored. The applied settings are adopted for the most part from the LNS parameters proposed in Parragh and Schmid (2013).

The results of our DP based hybrid LNS approach are shown in Table 2 for the a-instances, in Table 3 for the b-instances and in Table 4 for the pr-instances. The proposed approach is competitive with the state of the art in terms of solution quality. It is less than 1% worse on average for the a- and b-instances, but not more than 2.0% worse in the a-instances and 2.3% worse in the b-instances. While the run time for the two data sets of our framework is less. For the more difficult to solve pr-instances the results are on average 2.9% worse than the ones reported in Parragh and Schmid (2013). However, our approach was able to find a new best solution for instance pr12 and a better average solution could be found for instance pr11, showing the potential of our DP based hybrid LNS.

5 Conclusion

In this paper we have proposed dynamic programming based approaches for solving the dial-a-ride problem. These algorithms rely on a giant tour based representation of the DARP and are based on recent research on dynamic

			Literature			Restricted DP				Hybrid LNS			
Instance	m	n	Best	Avg.	CPU	Best	Avg.	CPU	%	Best	Avg.	CPU	%
b2-16	2	16	309.41	309.41	0.15	309.41	309.41	0.02	-0.0	309.41	309.41	0.06	0.0
b2-20	2	20	332.64	332.64	0.21	332.64	332.64	0.02	-0.0	332.64	332.64	0.10	0.0
b2-24	2	24	444.71	444.83	0.40	445.33	451.59	0.05	1.5	445.33	445.33	0.26	0.1
b3-24	3	24	394.51	394.51	0.31	395.05	403.31	0.07	2.2	394.51	394.51	2.64	0.0
b3-30	3	30	531.45	531.45	0.48	550.12	553.29	0.15	4.1	531.44	533.19	0.26	0.3
b3-36	3	36	603.79	604.13	0.74	631.69	642.87	0.25	6.4	606.20	607.33	0.34	0.5
b4-32	4	32	494.82	494.82	0.43	508.53	539.53	0.23	9.0	501.76	502.95	0.44	1.6
b4-40	4	40	656.63	656.63	1.00	665.54	665.54	0.33	1.4	658.18	660.99	0.69	0.7
b4-48	4	48	673.81	674.93	1.73	712.30	725.49	0.52	7.5	676.52	680.14	0.96	0.8
b5-40	5	40	613.72	613.73	0.78	630.39	656.91	0.37	7.0	614.94	615.40	0.77	0.3
b5-50	5	50	761.40	762.12	1.49	779.16	827.86	0.67	8.6	762.84	765.92	1.43	0.5
b5-60	5	60	902.52	903.46	3.00	971.92	1019.10	0.88	12.8	908.93	915.75	1.55	1.4
b6-48	6	48	714.83	714.83	1.07	783.27	789.85	0.54	10.5	718.82	719.19	1.33	0.6
b6-60	6	60	860.07	860.15	2.07	907.11	966.40	1.01	12.4	866.34	871.60	1.97	1.3
b6-72	6	72	979.61	980.27	4.43	1042.01	1134.45	1.44	15.7	988.63	995.17	4.13	1.5
b7-56	7	56	823.97	824.17	1.82	865.30	907.16	0.68	10.1	827.82	831.80	1.89	0.9
b7-70	7	70	913.11	914.97	3.70	953.70	1033.64	1.36	13.0	921.72	928.18	3.05	1.4
b7-84	7	84	1211.27	1213.55	6.72	1294.05	1392.00	2.13	14.7	1214.88	1233.71	4.65	1.7
b8-64	8	64	840.63	840.63	2.56	870.57	955.05	1.18	13.6	848.41	852.84	1.62	1.5
b8-80	8	80	1036.65	1038.28	3.84	1081.31	1176.74	1.83	13.3	1045.33	1051.80	2.00	1.3
b8-96	8	96	1187.80	1190.74	10.32	1244.60	1373.59	2.70	15.4	1205.05	1217.86	3.48	2.3
avg			727.97	728.58	2.25	760.67	802.69	0.78	8.5	732.37	736.46	1.60	0.9

Table 3 Results for the hybrid LNS and restricted DP algorithm for b-instances

			Literature			Restricted DP				Hybrid LNS			
Instance	m	n	Best	Avg.	CPU	Best	Avg.	CPU	%	Best	Avg.	CPU	%
pr01	3	24	190.02	190.02	0.54	190.49	209.60	0.10	10.3	190.02	190.30	0.75	0.1
pr02	5	48	301.34	302.53	2.76	317.09	353.50	0.54	16.8	301.34	304.13	9.28	0.5
pr03	7	72	535.28	538.21	5.11	594.81	620.20	1.14	15.2	538.73	545.80	10.56	1.4
pr04	9	96	571.09	576.26	16.29	678.69	706.14	2.40	22.5	585.77	599.87	8.54	4.1
pr05	11	120	629.52	637.59	26.70	702.04	765.63	3.59	20.1	651.20	659.72	41.22	3.5
pr06	13	144	788.88	800.35	48.48	934.07	959.43	5.19	19.9	822.52	837.95	24.06	4.7
pr07	4	36	291.71	292.56	1.02	308.21	320.66	0.27	9.6	291.71	296.12	1.94	1.2
pr08	6	72	491.93	495.20	5.92	558.41	576.40	1.09	16.4	503.17	508.28	8.79	2.6
pr09	8	108	661.47	676.09	24.89	823.44	836.08	2.39	23.7	691.80	717.62	13.92	6.1
pr10	10	144	872.31	878.93	47.17	988.26	1023.73	6.67	16.5	886.44	897.80	24.02	2.1
pr11	3	24	164.46	166.06	0.61	170.03	177.12	0.14	6.7	164.46	164.73	4.64	-0.8
pr12	5	48	295.96	298.88	3.00	322.13	334.39	0.57	11.9	**294.03**	**298.36**	32.34	-0.2
pr13	7	72	484.83	491.29	8.19	548.75	580.13	1.15	18.1	491.28	505.50	38.42	2.9
pr14	9	96	534.84	541.19	22.58	626.89	653.26	2.82	20.7	557.93	564.74	80.77	4.4
pr15	11	120	587.67	590.22	44.09	675.81	706.99	4.26	19.8	603.59	616.73	145.40	4.5
pr16	13	144	738.01	743.64	71.50	873.11	894.22	5.73	20.2	783.02	792.81	148.03	6.6
pr17	4	36	248.21	248.21	1.30	263.98	280.76	0.33	13.1	249.07	252.86	11.90	1.9
pr18	6	72	463.67	470.25	9.54	509.91	545.30	1.26	16.0	471.58	478.40	56.76	1.7
pr19	8	108	593.49	606.25	27.49	699.65	707.59	3.36	16.7	617.60	627.45	110.99	3.5
pr20	10	144	804.22	812.81	69.57	976.71	986.30	5.45	21.3	854.65	869.87	147.30	7.0
avg			512.45	517.83	21.84	588.12	611.87	2.42	16.8	527.50	536.45	45.98	2.9

Table 4 Results for the hybrid LNS and restricted DP algorithm for pr-instances

programming for vehicle routing problems Gromicho et al (2012). The main challenge was to incorporate user ride time constraints into the DP framework.

An exact DP algorithm has been developed, able to solve small benchmark instances to optimality. In order to be able to solve larger instances we propose a restricted DP algorithm. We studied several different selection functions allowing to determine which DP states survive in subsequent stages. The best results were obtained with a selection function not only taking account information about the current state, but also including estimations about the required travel distance to serve all open nodes and the average remaining time per open request. The DP approaches have then been combined with a large neighborhood search. In addition to existing destroy and repair operators, we introduced new DP based ones. There the idea of zero splits, as proposed in Parragh et al (2010b), is used to generate new *block-nodes* according to the resulting zero splits in a solution and solve a DP algorithm on these node sets again. To further destroy the solution, one operator set allows to decompose some of the *block-nodes* and use this as input for the DP algorithm while another operator selects tours from the solution and runs a restricted DP algorithm with the according requests and vehicles.

The computational results show that the proposed DP based hybrid LNS approach is competitive with the current state of the art. In terms of solution quality our approach is on average about 0.9% behind for the easier benchmark sets, while it requires about 30% less run-time on similar (slightly slower) hardware. For the harder benchmark instances our approach is worse on average, however it was able to find new best solution for one of the instances emphasizing the value and potential of our approach.

In current work we are incorporating the DP based approaches into a a real-world application of the dial-a-ride problem in a dynamic context. Additionally we plan to combine our the DP based approaches with other successful approaches from the literature.

Acknowledgments

This work, partially funded by the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT) within the strategic program FIT-IT ModSim under grant 822739 (project HealthLog).

References

Bellman R (1962) Dynamic programming treatment of the travelling salesman problem. J ACM 9:61–63

Bent R, Hentenryck PV (2006) A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. Computers & Operations Research 33(4):875 – 893, DOI 10.1016/j.cor.2004.08.001, Part Special Issue: Optimization Days 2003

Blum C, Puchinger J, Raidl GR, Roli A (2011) Hybrid metaheuristics in combinatorial optimization: A survey. Applied Soft Computing 11(6):4135 – 4151, DOI 10.1016/j.asoc.2011.02.032

Cordeau JF, Laporte G (2003) A tabu search heuristic for the static multi-vehicle dial-a-ride problem. Transportation Research Part B: Methodological 37(6):579 – 594

Cordeau JF, Laporte G (2007) The dial-a-ride problem: models and algorithms. Annals of Operations Research 153:29–46

Dumas Y, Desrosiers J, Gelinas E, Solomon MM (1995) An optimal algorithm for the traveling salesman problem with time windows. Operations Research 43(2):367–371

Funke B, Grünert T, Irnich S (2005) Local search for vehicle routing and scheduling problems: Review and conceptual integration. Journal of Heuristics 11:267–306

Gendreau M, Potvin JY, Brumlaysy O, Hasle G, Løkketangen A (2008) Metaheuristics for the vehicle routing problem and its extensions: A categorized bibliography. In: Golden B, Raghavan S, Wasil E (eds) The Vehicle Routing Problem: Latest Advances and New Challenges, Operations Research/Computer Science Interfaces Series, vol 43, Springer, pp 143–169

Gromicho JAS, van Hoorn JJ, Kok AL, Schutten JMJ (2012) Restricted dynamic programming: A flexible framework for solving realistic VRPs. Computers & OR 39(5):902–909

Held M, Karp RM (1961) A dynamic programming approach to sequencing problems. In: Proceedings of the 1961 16th ACM national meeting, ACM, New York, NY, USA, ACM '61, pp 71.201–71.204

Malandraki C, Dial RB (1996) A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem. European Journal of Operational Research 90(1):45 – 55

Norvig P (1992) Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp, 1st edn. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA

Parragh SN, Schmid V (2013) Hybrid column generation and large neighborhood search for the dial-a-ride problem. Computers & Operations Research 40(1):490 – 497

Parragh SN, Doerner KF, Hartl RF (2008) A survey on pickup and delivery problems. Part II. Journal für Betriebswirtschaft 58:81–117

Parragh SN, Doerner KF, Hartl RF (2010a) Demand responsive transportation. In: Cochran JJ, Cox LA, Keskinocak P, Kharoufeh JP, Smith JC (eds) Wiley Encyclopedia of Operations Research and Management Science, John Wiley & Sons, Inc., DOI 10.1002/9780470400531.eorms0243

Parragh SN, Doerner KF, Hartl RF (2010b) Variable neighborhood search for the dial-a-ride problem. Computers and Operations Research 37:1129–1138

Pisinger D, Ropke S (2010) Large neighborhood search. In: Gendreau M, Potvin JY (eds) Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol 146, Springer US, pp 399–419, DOI 10.1007/978-1-4419-1665-5_13

Psaraftis HN (1980) A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. Transportation Science 14(2):130–154

Ritzinger U, Puchinger J (2013) Hybrid metaheuristics for dynamic and stochastic vehicle routing. In: Talbi (2013), pp 77–95

Ropke S, Pisinger D (2006) An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science 40(4):455–472, DOI 10.1287/trsc.1050.0135

Ropke S, Cordeau JF, Laporte G (2007) Models and branch-and-cut algorithms for pickup and delivery problems with time windows. Networks 49(4):258–272

Schilde M, Doerner KF, Hartl RF (2011) Metaheuristics for the dynamic stochastic dial-a-ride problem with expected return transports. Computers & Operations Research 38(12):1719 – 1730

Shaw P (1998) Using constraint programming and local search methods to solve vehicle routing problems. In: Maher M, Puget JF (eds) Principles and Practice of Constraint Programming CP98, Lecture Notes in Computer Science, vol 1520, Springer Berlin Heidelberg, pp 417–431

Talbi EG (ed) (2013) Hybrid Metaheuristics, Studies in Computational Intelligence, vol 434. Springer Berlin Heidelberg

Toth P, Vigo D (eds) (2001) The vehicle routing problem. Society for Industrial and Applied Mathematics