Computers & Operations Research 40 (2013) 490–497

Contents lists available at SciVerse ScienceDirect

Computers & Operations Research

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

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

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

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

ARTICLE INFO

Available online 10 August 2012

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

ABSTRACT

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

1. Introduction

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

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

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

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

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

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

0305-0548/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cor.2012.08.004

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

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

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

2. Solution framework

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

2.1. Large neighborhood search

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

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

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

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

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

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

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

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

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

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