Online Dial-A-Ride Problem with Time Windows: An Exact Algorithm Using Status Vectors

Anke Fabri and Peter Recht

Universität Dortmund, Germany
a.fabri@wiso.uni-dortmund.de
p.recht@wiso.uni-dortmund.de

Summary. The Dial-A-Ride Problem (DARP) has often been used to organize transport of elderly and handicapped people, assuming that these people can book their transport in advance. But the DARP can also be used to organize usual passenger or goods transportation in real online scenarios with time window constraints. This paper presents an efficient exact algorithm with significantly reduced calculation times.

1 Introduction

The problem consists in routing a vehicle that serves a set N of demands growing in time. The road network is represented by a weighted graph $G = (V,E,d)$ whose vertices $v \in V$ correspond to the possible pickup and delivery points and whose arcs $e \in E$ represent shortest paths between these points. The cost of each arc is given by a function d: $E \rightarrow \Re^+$, indicating the length of the shortest path. The planning horizon is denoted by the interval $[0,T]$. Demands for transportation pop up dynamically, without any information in advance about time of appearance, location or quantity of incoming demands. Each demand $i \in N$ consists of a pickup point $u_i \in V$ with a pickup time window $[a_i, b_i] \subset [0, T]$, a delivery point $v_i \in V$ with a delivery time window $[c_i, e_i] \subset [0,T]$ and the quantity of goods to be transported. The vehicle is characterized by its starting point s where it is located at time 0, its end point z where it should be at time T and its capacity k. When at any time a new demand pops up, it must be decided immediately whether the demand is accepted or rejected. If it is accepted, a valid tour has to be found that serves all accepted demands meeting all time windows exactly. The vehicle may wait for service. The objective consists of two parts: first, as many demands as possible shall be served. Second, the traveled distance needed to serve all accepted demands shall be minimized.
This paper presents a special graph structure to handle the DARP and sketches the solution algorithm. The results show that calculation efforts and therefore calculation times are reduced significantly.

2 The Status Vector Graph

The basic status vector graph $G'_n = (V'_n, E'_n, d')$ forms the basis for a dynamic programming solution sequencing the pickup and delivery points of all n accepted demands. Each accepted demand $i \in N$ is characterized by its status $j(i) \in \{0,1,2\}$. Here, $j(i) = 0$ means 'demand i is still waiting for service', $j(i) = 1$ denotes 'demand i has been picked up, but not yet delivered' and $j(i) = 2$ says 'demand i has been served completely'. A status vector is a vector $(j(i_1),j(i_2),\dots j(i_n))$ that represents the status of each demand $i \in N$ with $\{i_1,i_2,\dots i_n\} = N$. The set of all possible status vectors of all demands $i \in N$ forms the set of vertices V'_n. Two vertices $v', v'' \in V'_n$ are connected by an edge $(v',v'') \in E'_n$ whenever vector v'' can be obtained from vector v' by adding 1 to exactly one vector element of v'. This change corresponds to a pickup or delivery of one demand, implying a vehicle movement between two nodes in G. The arc (v',v'') is therefore weighted with the corresponding costs (see [1]). Note that the single status vector does not contain any information about the vehicle's location. The source vertex is the status vector representing the actual status when the calculation is started, the sink vertices are $(2,\dots,2)$. Figure 1

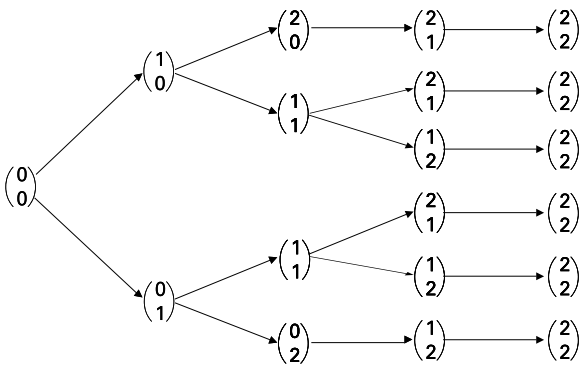

Fig. 1. Basic Status Vector Graph for two Demands

shows G'_2 for both demands still waiting for pickup. Calculating a tour now consists in finding a shortest path from the source to one of the sink vertices subject to time window and capacity constraints.

Attention is attracted by the fact that many status vectors are identical. Calculation times could be significantly reduced, if these multiple status vectors could be merged to obtain a graph of reduced size, say ${}_{red}G'_n = ({}_{red}V'_n, {}_{red}E'_n, {}_{red}d')$. Due to the cost function depending on the vehicles location, two identical vectors can only be merged if also the vehicle location is identical. Hence, one more piece of information has to be stored. See figure 2 for an example of ${}_{red}G'_2$ with pickup places A, B and delivery places C, D for demand $1, 2$, both still waiting for service.

Regarding the number of nodes of ${}_{red}G'_n$, one must state that it is worth storing the location additionally: