The Truckload Trip Scheduling Problem

Claudia Archetti
Department of Quantitative Methods, University of Brescia
Contrada Santa Chiara 50, 25122 Brescia, Italy

Martin Savelsbergh
School of Industrial and Systems Engineering, Georgia Institute of Technology
Atlanta, GA 30332-0205, U.S.A.

1 Introduction

Truckload transportation represents a significant portion of all land-based freight transportation. In truckload transportation full truckloads have to be picked up at an origin location and delivered at a destination location. A load dispatch window specifies the earliest time a load is ready for pickup at the origin location and the latest time a load can be picked up at the origin location and still reach the destination location at or before a desired time. The challenge in truckload transportation is minimizing the (empty) repositioning of vehicles between a delivery location and a subsequent pickup location, because no revenues are generated on such repositioning moves, only costs are incurred. Demand uncertainty (transportation requests become available dynamically over time) makes minimizing repositioning especially challenging. A body of literature exist on truckload transportation, see for example [2, 3, 4, 5, 6, 7]. Unfortunately, in almost all of the literature one essential real-life complexity is ignored: driver restrictions. Governments impose restrictions on truck drivers to ensure their safety as well as the safety of other drivers. In the United States, for example, the Department of Transportation (DOT) mandates that a driver cannot drive for more than 11 hours and cannot be on duty for more than 14 hours before a mandatory rest of at least 10 hours has to be taken (there are additional restrictions, but these two are the most important). It is obvious, especially because truckload transportation typically involves moving loads over long distances, that decision technology for scheduling truckload transportation requests that ignores driver restrictions is of limited, if any, value. One of the few papers explicitly acknowledging the importance and complexity of handling driver restrictions is Xu et al. [1]. The paper contains illustrative examples of how DOT rules impact the timing of a trip (i.e., arrival and departure times at pickup and delivery locations). The authors observe that due to the presence of dispatch windows at origin locations of loads and DOT rules governing drivers it is not necessarily optimal for a truck to depart from its home base as early as possible in order to complete a trip as early as possible. In fact, they conjecture that this "trip scheduling" problem is NP-hard. Our research disproves that conjecture and shows that this trip scheduling problem can be solved in polynomial time.

2 Problem Definition

Truckload transportation problems are characterized by a set of transportation requests I, with for each transportation request $i \in I$ an origin location i^+, a destination location i^-, and a dispatch window at the origin location $[e_{i^+}, \ell_{i^+}]$. If a truck arrives at an origin location before the opening of the dispatch window, it has to wait. A truck is not allowed to arrive at an origin location after the closing of the dispatch window. A truck can serve one transportation request at a time. For convenience, we assume that the pickup and the delivery of a load happen instantaneous (i.e., loading and unloading times are ignored). A truck driver can drive at most τ_{drive} hours and can work (be on duty) at most τ_{duty} hours before a mandatory rest of τ_{rest} hours.

A trip serving a set of transportation requests S, with $|S| = n$, starts and ends at a central facility (with location 0) and serves the transportation requests in S in a specified sequence. The *Truckload Trip Scheduling Problem* (TSPP) is to determine whether or not a departure time at the central facility and a set of dispatch times at the origin locations of the transportation requests exists that results in a feasible schedule.

Since the sequence of transportation requests is given and there are no time restrictions at the delivery locations, it is possible to focus completely on dispatch times at the origin locations of the loads. Once a load departs at its origin location, the arrival time at the origin location of the next load is completely determined (travel time to the destination location plus the relocation time from the destination location to the origin location of the next load).

3 Algorithm

The key component of our algorithm is a backward search through the trip establishing either a feasible dispatch schedule or identifying a "difficult" dispatch window. Assume that the trip is given by a sequence of locations $(0, 1, 2, ..., n, n+1)$, where 0 and $n+1$ represent the starting and ending location. The backward search finds a path from $n+1$ to i for each location i starting at time l_{i+1} at $n+1$ with a minimum number of rests and maximum remaining duty time at i. The backward search is described precisely in Algorithm 1, where *remDuty* keeps track of the remaining duty time, *remDrive* keeps tracks of the remaining drive time, and *flexTime* keeps track of the amount of time by which the path can feasibly be "shifted to the left" without affecting the remaining duty and drive time (it captures the flexibility resulting from the width of the dispatch windows as well as the flexibility resulting from the difference between τ_{duty} and τ_{drive}).

When the algorithm stops prematurely, i.e., without establishing a feasible dispatch time at every location, we have identified a location i for which the constructed path does not provide a time feasible path from i to $n+1$ (as we would have to depart before the opening of the dispatch window). The constructed path is made up of drive time, waiting time, and rest time. Because the path has the minimum number of rests, a feasible path from i to $n+1$ has to have less waiting time. (Note: drive time and rest time cannot be reduced.) The only way to reduce waiting time is to convert waiting time into rest time by taking rests later in time ("pushing rests up"). In fact, rests can only be pushed up since, by construction on the solution, they are already made as late as possible so they can not be "pushed down." An example of reducing waiting time by taking rests later is depicted in Figure 1. Instead of taking the rest between the first and second location, the rest is taken later at the second location.

The only place where waiting time is introduced into the constructed path in Algorithm 1 is at

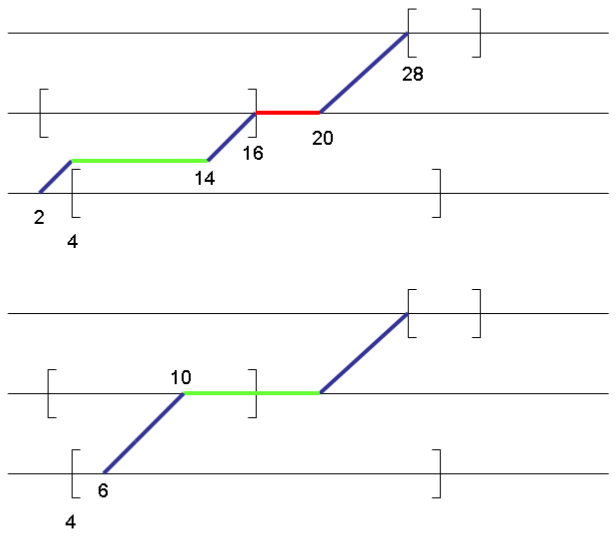

Figure 1: Converting waiting time to rest time

line 26; the driver waits $t - l_i$. This allows us to keep track not only of the total waiting time on the path, but also of the locations where we may be able to convert waiting time to rest time. To restore feasibility, we push rests up so as to convert enough waiting time into rest time to restore feasibility at i.

This suggests Algorithm 2 for determining a set of feasible dispatch times at each location or establishing that no feasible schedule exists. If the backward search identifies a "difficult" dispatch window, a forward search is started to push up rests and to convert enough waiting time into rest time to recover feasibility.

The forward search examines nodes along the path to $n+1$ in order until a node with waiting is reached. When a node with waiting time is encountered, either there is no rest that can be pushed up to convert this waiting time into rest time, in which case there exists no feasible solution (because the path already arrives at i as late as possible) or there is a rest that can be pushed up to convert this waiting time into rest time, in which case the rest *has* to be pushed up and the waiting time *has* to be converted into rest time (otherwise the time at node i will not change). If the amount of waiting time converted to rest time is insufficient to restore feasibility, then the forward search continues.

A key observation is that *after a rest has been pushed up, it can never be pushed down, because that would reintroduce infeasibility.* As a result, the algorithm is polynomial. At most n feasibility recovery actions are need (there are only n locations on the path), and each feasibility recovery action involves no more than n wait-to-rest conversions. This shows that the run time of the algorithm is $O(n^2)$.

4 Future Extensions

Several variants of the Truckload Trip Scheduling Problem can be defined by changing some of the assumptions, for example by assuming fixed dispatch times or by assuming that the rest time is at least Δ_{rest} as opposed to exactly Δ_{rest}. These variants are under investigation.

References

[1] H. Xu, Z.-L. Chen, S. Rajagopal, and S. Arunapuram. "Solving a Practical Pickup and Delivery Problem." *Transportation Science 37,* 347-356, 2003.

[2] W.B. Powell. "A Stochastic Model of the Dynamic Vehicle Allocation Problem," *Transportation Science 20,* 117-129, 1986.

[3] W.B. Powell. "An Operational Planning Model for the Dynamic Vehicle Allocation Problem with Uncertain Demands," *Transportation Research 21B,* 217-232, 1987.

[4] W.B. Powell, A. Marar, J. Gelfand, S. Bowers. "Implementing Real-Time Optimization Models: A Case Application From The Motor Carrier Industry." *Operations Research 50,* 571-581, 2002.

[5] J. Yang, P. Jaillet, and H. Mahmassani. "Real-Time Multi-vehicle Truckload Pickup and Delivery Problems." *Transportation Science 38,* 135-148, 2004.

[6] A. Regan, H. Mahmassani, and P. Jaillet. "Evaluation of Dynamic Fleet Management Systems." *Transportation Research Record 1645,* 176-184, 1998.

[7] W.B. Powell, Y. Sheffi, K. Nickerson, K. Butterbaugh, and S. Atherton. "Maximizing Profits for North American Van Lines' Truckload Division: A New Framework for Pricing and Operations," *Interfaces 18,* 21-41, 1988.

Algorithm 1 Backward Search Algorithm

```
1: $t \leftarrow l_{n+1},\ remDuty \leftarrow \tau_{duty},\ remDrive \leftarrow \tau_{drive},\ flexTime \leftarrow \infty$
2: for $i = n, n-1, \ldots, 1$ do
3:   if $t_{i,i+1} < remDrive$ then
4:     $t \leftarrow t - t_{i,i+1},\ remDrive \leftarrow remDriver - t_{i,i+1},\ remDuty \leftarrow remDuty - t_{i,i+1}$
5:   else
6:     $nRests \leftarrow 1 + \lfloor \frac{t_{i,i+1} - remDrive}{\tau_{drive}} \rfloor$
7:     $t \leftarrow t - (t_{i,i+1} + nRest \times restTime)$
8:     $remDrive \leftarrow \tau_{drive} - (t_{i,i+1} - remDrive) \mod \tau_{drive}$
9:     $remDuty \leftarrow \tau_{duty} - (t_{i,i+1} - remDrive) \mod \tau_{drive}$
10:  end if
11:  if $t < e_i$ then
12:    Stop // Dispatch window violation
13:  else
14:    $additionalFlexTime \leftarrow (remDuty - remDrive) + (nRest - 1) \times (\tau_{duty} - \tau_{drive})$
15:    $flexTime \leftarrow flexTime + additionalFlexTime$
16:    if $e_i \leq t \leq l_i$ then
17:      $flexTime \leftarrow \min(flexTime, t - e_i)$
18:    else
19:      $tBest \leftarrow t - (flexTime + remDuty)$
20:      if $tBest > l_i$ then
21:        $nRests \leftarrow \lceil \frac{tBest - l_i}{restTime} \rceil$
22:        $t \leftarrow t - nRests \times restTime$
23:      end if
24:      if $t - l_i > flexTime$ then
25:        $t \leftarrow t - flexTime,\ flexTime \leftarrow 0$
26:        $remDuty \leftarrow remDuty - (t - l_i)$
27:        $remDrive \leftarrow \min(remDrive, remDuty)$
28:      else
29:        $flexTime \leftarrow \min(flexTime - (t - l_i), l_i - e_i)$
30:      end if
31:      $t = l_i$
32:    end if
33:  end if
34: end for
```

Algorithm 2 Algorithm to identify feasible schedule

```
1: $Done \leftarrow False$
2: repeat
3:    Apply BackwardSearch
4:    if $feasible\ schedule\ identified$ then
5:       $Done \leftarrow True$ // Feasible schedule exists
6:    else
7:       Apply ForwardSearch // Move rests up to restore feasibility
8:       if $Feasibility\ is\ not\ recovered$ then
9:          $Done \leftarrow True$ // No feasible schedule exists
10:      end if
11:   end if
12: until Done
```