An Online Schedule Update Algorithm for Vehicle Fleets
CMIS Technical Report Number 04/17

Philip Kilby

CSIRO Mathematical and Information Sciences,
GPO Box 664, Canberra ACT 2601, Australia
Phil.Kilby@csiro.au
January 2004

Abstract

The problem studied here is that of maintaining an efficient assignment of recreational rental vehicles to a set of bookings. An on-line system called D-VASS has been developed which maintains a near-optimal assignment of vehicles to bookings, subject to the client's business rules.

The system operates in a real-time environment. Operations such as replying to availability queries, incorporating a new booking into the schedule, and recovering after unplanned schedule disturbances must be completed very quickly.

This paper describes the scheduling problem faced, and the algorithms employed by D-VASS in order to maintain the schedule.

1 Introduction

The problem studied here is that of eating and maintaining an efficient assignment of recreational rental vehicles to a set of bookings.

Tourism Holdings Limited (THL) run a fleet of around 4000 *recreational rental vehicles* of many types in Australia and New Zealand. Vehicles in a dozen locations are scheduled centrally by a system called D-VASS. Assigning a vehicle to each booking produces a vehicle schedule. If demand outstrips supply at one location, vehicles can be relocated, or substitutions of vehicle types can be made. Each booking can be therefore be covered by a potentially large number of possible vehicles. There can be a very large difference in cost between a good schedule and a poor schedule. The schedule also determines vehicle availability for a potential new booking. If the new booking cannot be incorporated into the schedule, this may result in lost business for THL. Schedule production is therefore a core business activity.

There are two parts to scheduling at THL. The first is to produce a schedule for the bookings already accepted. We call this the *static* scheduling problem. The second part of the problem is to maintain the schedule during the day, as new bookings are accepted, and day-to-day disruptions such as vehicle break-downs occur. This part of the problem is called the *dynamic* scheduling problem.

CSIRO has written a system called VASS that solves the static scheduling problem at THL. VASS is run each night, and produces a near-optimal schedule incorporating all (or as many as possible) of the bookings already accepted. VASS is not described further in this paper.

CSIRO has also supplied a system – called D-VASS – to solve the dynamic scheduling problem. D-VASS lies at the heart of the reservations and operations systems at THL. D-VASS starts with the solution to the static problem. All queries on vehicle availability for a new booking, all new bookings, plus all disruptions to the schedule are handled by D-VASS. D-VASS attempts to maintain a near-optimal schedule as events unfold over the course of a day.

This paper describes the primary algorithm used by D-VASS in order to maintain the schedule.

The algorithm is used in four contexts:

- to answer availability queries,
- to incorporate a new booking into the schedule,
- to repair the schedule after a disturbance, and
- in a background "improvement" process run while the system is quiescent.

The application of the algorithm in each of these contexts will be examined in turn.

The D-VASS algorithm is based on Engquist's Successive Shortest Path Algorithm for the Assignment Problem ([5]). As Engquist's algorithm is important to understanding the current work, it is reviewed in Section 3 below.

In addition to satisfying current and incoming bookings, a number of regular operational requirements must be satisfied. Vehicles must be scheduled for maintenance at specified depots. Selected vehicles must be removed from the fleet as their useful life comes to an end. Unlike bookings, maintenance and disposal activities usually involve a specific nominated vehicle (bookings only refer only to a product and hence, a preferred vehicle *type*).

The extra constraint on disposals and maintenance to use a particular vehicle does not fit within the model presented in this paper. They are handled by special code in D-VASS. For ease of explanation, they are omitted from this description of the problem.

A more complete description of scheduling at THL is given in Section 2. We describe how the basic algorithm is used in these contexts in Sections 4.1 and 4.2. We conclude with a discussion of the performance of the system in Section 5, and directions for future work in Section 6.

2 The Scheduling Problem

THL operate a fleet of more than 2000 recreational vehicles in New Zealand, and approximately the same size fleet in Australia. There are about 50 distinct vehicles types, ranging from a standard 4-door saloon car, through 2-berth and 4-berth vehicles to 6-berth camper-vans, and including a range of optional extras.

Because of the nature of the recreational vehicle market, THL often see long booking horizons and long booking durations; it is not unusual for bookings to be made one year in advance. A long scheduling and planning horizon is therefore required when putting together a schedule. THL use an *active horizon* of 300 days, which currently includes more than 7,000 bookings in New Zealand, and 10,000 in Australia. Bookings beyond 300 days are automatically accepted, but do not appear on the schedule.

A basic concept of the system is the *booking*. When making a booking, customers specify their request in terms of *products*. Each product is characterized by a preferred vehicle type, but also has a ranked list of acceptable substitutes. The other important data for a booking are the date and location the vehicle will be picked up, and the date and location it will be returned.

THL allow vehicles to be checked out at one city but returned to another, at no extra cost to the customer. This leads to significant complications for THL's scheduling staff. Unlike the situation in most rental car hire companies, vehicles are not owned by individual branches. Consequently there is no need for the vehicles to return to their point of origin. Under these circumstances, there are invariably imbalances between the number of vehicles leaving and returning to a particular location. This requires vehicles to be regularly *relocated* from one vehicle depot to another in order to satisfy bookings. Alternatively, a *substitute* vehicle (usually of higher commercial value) may be offered to satisfy a given bookings. Vehicle substitutions are assigned a notional (non-negative) cost, comparable to a dollar cost, that indicates the expected level of customer dissatisfaction at such a substitution.

An effective schedule must minimize the number, the notional cost, and the dollar cost of relocations and substitutions in order to maximize both profit and customer satisfaction.

Given that vehicles may be relocated or products substituted, even the question "is there a vehicle available to cover a booking for product X?" does not have an easy answer. In a standard vehicle rental operation where a significantly large proportion of vehicles return to the point of origin, it is fairly easy to answer the above question by merely counting the number of bookings that have to be catered for on each day. If there are fewer bookings than vehicles in the fleet, then it can be assumed that a vehicle will be found to satisfy the booking request. However, at THL a plethora of relocation and substitution options are available and these must be considered when evaluating availability.

The question of availability must typically be solved in near-real time, while an operator replies to a customer who has called to make a booking enquiry.

The dynamic scheduling problem addressed here refers also to the smooth handling of every-day, unplanned disruptions to the schedule, including the late return of a vehicle, vehicle break-downs, and incorrect dispatch by THL staff (e.g. when a customer is given the wrong vehicle)

The cost of a solution in the D-VASS algorithm is the sum of the individual costs of vehicle assignments. The cost of assigning a booking to a particular vehicle is a combination of many factors. These include

- A penalty associated with any substitution (e.g. using a 4-berth vehicle for a 2-berth booking). This is a function of the product specified in the booking, and the type of the vehicle assigned.
- The cost of relocating the vehicle from the return point of the previous booking. Relocations of various durations are allowed, at different costs.
- The *turnaround* penalty. The turnaround is the time available to clean and prepare a vehicle after its previous booking. A penalty is applied to shorter turnarounds to encourage more time to be allowed for cleaning.
- A delay penalty. A delay, where a customer is forced to wait for one half or one day, is allowed but heavily penalized. D-VASS plans for delay only in emergency situations, such as breakdown of the only vehicle available for a given booking.

3 Engquist's Algorithm

This section gives a brief, and informal description of Engquist's algorithm. Some changes to the algorithm have been made for ease of explanation. For a full treatment, see the original work [5].

The Assignment Problem (also known as the bipartite matching problem) can be stated as follows. Consider sets I and J with $|I| = |J|$. For each $i \in I$ and $j \in J$ there is an assignment cost c_{ij} (c_{ij} may be infinite if the assignment is not permitted). We wish to find a set of assignments $A \subset I \times J$, pairing each i and j which minimizes the assignment cost. In Linear Programming terms, define $x_{ij} = 1$ (for $i \in I$ and $j \in J$) if assignment $i \to j$ is in the solution, 0 otherwise. Then

$$\text{minimize} \quad \sum_{i \in I} \sum_{j \in J} x_{ij} c_{ij} \tag{1}$$

$$\text{subject to} \quad \sum_{i \in I} x_{ij} = 1 \quad \forall j \in J \tag{2}$$

$$\sum_{j \in J} x_{ij} = 1 \quad \forall i \in I \tag{3}$$
$$x_{ij} \in \{0,1\} \quad \forall i \in I, \forall j \in J \tag{4}$$

Enquist's algorithm is essentially a dual algorithm. It relaxes the constraint 3 that each j is matched to exactly one i. It then successively tightens this relaxation.

It begins by assigning every i its cheapest j. After this, it is typically the case that some j are *deficient* (no i assigned) and some *abundant* (more than one i assigned). Each step of the algorithm reduces the number of deficient nodes, eventually resulting in a solution to the original problem.

The best way to see how the algorithm works is to look at an example iteration. Consider $I = \{a, b, c, d, e\}$ and $J = \{A, B, C, D, E\}$. At the current step of the algorithm, the assignments are

$$\begin{array}{ccc} a & \rightarrow & A \\ b & \rightarrow & A \\ c & \rightarrow & C \\ d & \rightarrow & B \\ e & \rightarrow & B \end{array}$$

So A and B are abundant, while D and E are deficient.

The algorithm operates using a *meta-graph*. There is a meta-node in the graph representing each current assignment. In our example there would be five meta-nodes: $(a \rightarrow A)$, $(b \rightarrow A)$, $(c \rightarrow A)$, $(d \rightarrow B)$, and $(e \rightarrow B)$.

A special, dummy meta-node is also added corresponding to an arbitrary deficient node. In our example, we choose E as the deficient node, and add the meta-node $(dummy \rightarrow E)$.

The meta-graph has arcs corresponding to "swaps" in the assignment. The intuitive interpretation of an arc from $(i \rightarrow j)$ to $(m \rightarrow n)$ is: delete assignment $(i \rightarrow j)$ and replace it with assignment $(m \rightarrow j)$. An arc from $(i \rightarrow j)$ to $(m \rightarrow n)$ exists if c_{mj} is finite. The cost of this arc is c_{mj} - c_{ij}.

The algorithm finds the shortest path within the meta-network from the dummy node to each abundant meta-node (that is, a meta-node representing an assignment to an abundant node). It then chooses the least-cost shortest path, and implements the "swaps" implied in that path.

For example, suppose that in the above example the least cost path is

$$\begin{array}{cccc} (dummy & \rightarrow & E) & \Rightarrow \\ c & \rightarrow & C & \Rightarrow \\ b & \rightarrow & A & \end{array}$$

This path results in the assignments along the path being replaced with

$$\begin{array}{ccc} c & \rightarrow & E \\ b & \rightarrow & C \end{array}$$

The new assignments are

$$\begin{array}{ccc} a & \rightarrow & A \\ b & \rightarrow & C \\ c & \rightarrow & E \\ d & \rightarrow & B \\ e & \rightarrow & B \end{array}$$

Now only B is abundant, and only D deficient. Because at each step the number of deficient nodes strictly decreases by one, the algorithm finishes in a finite number of steps when no more deficient nodes exist.

Engquist uses the all-pairs shortest path algorithm of Dial [3] to find the shortest path in the meta-network, but any shortest path algorithm can be used.

4 The Basic Algorithm

This section describes the use of an adaptation of Engquist's algorithm for use on-line.

The current schedule of bookings on vehicles can be represented as an assignment of each booking to its successor (i.e. the next booking on that vehicle), that is an assignment of an activity $i \in \mathcal{I}$ to its successor activity $j \in \mathcal{J}$.

The set $\mathcal{I}$ includes all bookings, plus a number of activities representing the availability of a vehicle – the *start activity*. We know where and when each vehicle will become available again (when it will be returned).

The set $\mathcal{J}$ also includes a node for each booking, plus an *idle activity* for each vehicle – the activity undertaken after all bookings have been completed. The idle activity can be carried out anywhere at any time.

For example, a schedule with Vehicle 123 doing bookings A, B, and C would be represented as

$$\begin{array}{rcl} \textit{Start-123} & \rightarrow & A \\ A & \rightarrow & B \\ B & \rightarrow & C \\ C & \rightarrow & \textit{Idle-123} \end{array}$$

The cost c_{ij} of linking two activities depends only on the time and location of the completion of activity i, and the time and location of the start of activity j. If i ends at a location distinct from the location at which j starts, then a relocation is required. The cost of the relocation can be calculated given the two end-points, and the time available. If j starts before i finishes, either a delay is required and can be costed, or the connection is illegal and the link is not added. Other costs, such as the turnaround cost discussed in Section 2, can also be calculated given the information available.

4.1 Adding a new booking

Queries and new bookings are handled in the same way. For a query, the solution is discarded once the existence of a solution is established. When adding a new booking, the updated schedule becomes the new incumbent schedule. Call the new booking Q.

The algorithm proceeds by looking at each possible vehicle type t that can be used for Q. The algorithm examines how Q might be inserted into the sub-schedule representing only vehicles of type t.

The total cost for using vehicle type t is the substitution cost of using t for Q, plus the incremental cost of inserting Q into the sub-schedule. The algorithm then chooses the schedule with the least total cost.

The following describes how Q is inserted into the sub-schedule of vehicles of type t.

We examine the current assignments for vehicle type t. Let the existing assignment (successor) of booking i be denoted S_i. As in the Engquist algorithm, create a meta-network with nodes representing the current assignments.

The arcs of the meta-network are as for the Engquist algorithm. In the D-VASS context, many arcs will be illegal. The arc from meta-node $(i \rightarrow S_i)$ to $(j \rightarrow S_j)$ will only be legal if S_i is a legal successor for j. This will only be possible if S_i occurs later than j; if the relocation implied (if any) is legal; if the time allowed for turnaround is legal; etc.

We add two dummy meta-nodes $(dummy \rightarrow Q)$, $(Q \rightarrow dummy)$. In effect, *dummy* is abundant (it has one more assignment than we want). If we find the shortest path from $(dummy \rightarrow Q)$ to $(Q \rightarrow dummy)$ we can incorporate Q into the schedule optimally (using a similar argument to that used in Engquist).

For example, the path

$$\begin{array}{rcll} (dummy & \rightarrow & Q) & \Rightarrow \\ (i & \rightarrow & S_i) & \Rightarrow \\ (j & \rightarrow & S_j) & \Rightarrow \\ (Q & \rightarrow & dummy) & \end{array}$$

would result in the updated assignments

$$\begin{array}{rcl} (i & \rightarrow & Q) \\ (j & \rightarrow & S_i) \\ (Q & \rightarrow & S_j) \end{array}$$

If the original schedule was
$i \rightarrow i_2 \rightarrow i_3 \rightarrow i_4$
$j_0 \rightarrow j \rightarrow j_2 \rightarrow j_3$
then the new schedule would be
$i \rightarrow Q \rightarrow j_2 \rightarrow j_3$
$j_0 \rightarrow j \rightarrow i_2 \rightarrow i_3 \rightarrow i_4$

It is clear that very radical changes to the schedule can be made using this technique.

The technique is well suited to operation with limited execution time. The paths originating at $(dummy \rightarrow Q)$ are explored in a breadth-first manner (all paths of length 2 are explored before moving on to paths of length 3). Any path from $(dummy \rightarrow Q)$ to $(Q \rightarrow dummy)$ represents a feasible update, and can be implemented to incorporate Q. However if the path found within the limited execution time is not, in fact, the shortest path, then the updated schedule will not be optimal. In the D-VASS context, a near-optimal solution is quite acceptable.

The fact that the original schedule may not be optimal has implications for the shortest path algorithm. A sub-optimal schedule will contain cycles in the meta-network with a negative cost - that is, will contain a path from some node back to itself with a negative cost. The existence of negative-cost cycles means that many standard shortest-path algorithms (such as Dijkstra's algorithm [4]) cannot be used. Instead, the Bellman-Ford algorithm [1, 6] is used in D-VASS, as implemented in Cormen *et al* [2].

If, during the search for a path in the meta-network, a negative-cost cycle is found, it is simply implemented, the meta-network repaired, and the search re-started.

4.2 Repairing and improving the schedule

As the day progresses, the sub-optimal elements may be introduced into the schedule. There are several possible causes. The most common is that an update to the schedule was curtailed by limited execution time. For example, a new booking was added to the schedule, but the optimal position was not found within the execution time available. Another cause is when a disruption to the schedule occurs, such as a late return of a vehicle. The schedule may still be feasible, but a different solution may have a better objective value.

This section describes how these sub-optimal elements are repaired in the D-VASS algorithm.

As discussed in Section 4.1, an improvement to the schedule appears as a negative-cost cycle in the meta-network. This fact is used to drive both the repair and continual-improvement algorithms.

The continuous-improve function first creates the meta-network as in Section 4.1. It then chooses a random, existing assignment, and begins building the shortest path tree in the meta-network rooted at the meta-node representing the chosen assignment. The idea is to discover a negative-cost cycle. The chosen node itself does not have to be part of the cycle – as the shortest path tree extends out from the chosen node, cycles in other parts of the meta-network will be discovered. As negative-cost cycles are discovered, they are implemented.

This improve function can be called at any time on any part of the schedule to try to find an improvement in that area of the schedule.

Now consider the task of repairing the schedule after a disruption, such as a late return of a vehicle. This results in potential conflicts with existing bookings. Again the usual meta-network is created. Instead of choosing a random node, however, the shortest path tree is built around a node close to the disruption. In this way, that part of the network is more thoroughly explored, and useful repairs are more likely to be discovered.

In both these cases, limited CPU time is allocated. Neither method is guaranteed to find optimal updates.

5 Performance

The target platform for D-VASS was Microsoft Windows 2000. The D-VASS algorithm is written in C++, wrapped in a DLL, and further wrapped as a Microsoft COM+ server. This allows it to be called from Microsoft Visual Basic, the language used for the rest of THL's systems.

D-VASS operates 23.5 hours per day, seven days a week. It runs on a 1.5GHz Pentium PC under Microsoft Windows 2000.

User parameters set the available time for completing a call. A short time may lead to D-VASS being unable to find a find an implementation for a requested booking, even though one exists. This is a poor result as it may lead to lost business for THL. On the other hand, making a customer wait for too long for an answer can also lead to customer dissatisfaction.

Currently an availability or add-rental call must be completed within two seconds. This is seen as an acceptable trade-off between delaying the customer, and finding a reasonable solution.

Repairing a schedule after disruption is also limited to two seconds on-line.

The improvement process runs continuously when there is no other activity on the server. Any improvements detected are implemented immediately. The improvement process is cancelled as soon as another call is made on the system.

Using these settings, the dynamic system is able to maintain a solution quality similar to that produced by the overnight, near-optimal run. That is, the objective value at the end of the day is consistently within a few percent of the value subsequently produced by the overnight run.

6 Conclusions

CSIRO Mathematical and Information Sciences has produced an on-line scheduling system to assist a large recreational vehicle rental company in their day-to-day operations.

The software is able to untangle the very difficult problems that such a company must solve repeatedly:

- Do I have a vehicle available to cover a potential booking?
- When should I use substitutions rather than relocations?
- How do I adjust my schedule to cope with a disruption?

Good answers to these questions are central to the efficient running of the company.

The software uses an on-line update algorithm based on successive shortest paths to maintain and improve the schedule. The algorithm has been described in detail in this paper.

The software sits at the heart of the reservations and operations systems of the company, and maintains an efficient vehicle schedule throughout the day.

7 Acknowledgements

Although the algorithm described here was developed by the author, the construction of the VASS and D-VASS software systems was very much a team effort. The other team members were Andreas Ernst, Mark E.T. Horn and Mohan Krishnamoorthy. Each member of the team has contributed substantially to the success of the project.

References

[1] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[2] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Electrical Engineering and Computer Science Series. The MIT Press, Cambridge Massachusetts, 1989.

[3] Robert B. Dial. Algorithm 360: Shortest-path forest with topological ordering. *Comm. of the ACM*, 12(11):362–363, November 1969.

[4] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[5] Michael Engquist. A successive shortest path algorithm for the assignment problem. *Infor*, 20(4):370–384, November 1982.

[6] Lestor R Ford Jr. and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.