Production, Manufacturing and Logistics

A multi-commodity, capacitated pickup and delivery problem: The single and two-vehicle cases

Harilaos N. Psaraftis*

Laboratory for Maritime Transport, National Technical University of Athens, Greece

A R T I C L E  I N F O

Article history:
Received 15 December 2010
Accepted 28 June 2011
Available online 6 July 2011

Keywords:
Routing
Scheduling
Pickup and delivery

A B S T R A C T

We explore dynamic programming solutions for a multi-commodity, capacitated pickup and delivery problem. Cargo flows are given by an origin/destination matrix which is not necessarily symmetric. This problem is a generalization of several known pickup and delivery problems, as regards both problem structure and objective function. Solution approaches are developed for the single-vehicle and two-vehicle cases. The fact that for each cargo that goes from a node $i$ to another node $j$ there may be a cargo going in the opposite direction provides the motivation for the two-vehicle case, because one may conceivably consider solutions where no cargoes that travel in opposite directions between node pairs are carried by the same vehicle. Yet, it is shown that such scenarios are generally sub-optimal. As expected, the computational effort of the single vehicle algorithm is exponential in the number of cargoes. For the two-vehicle case, said effort is of an order of magnitude that is not higher than that of the single-vehicle case. Some rudimentary examples are presented or both the single-vehicle and two-vehicle cases so as to better illustrate the method.

© 2011 Elsevier B.V. All rights reserved.

1. Introduction

We are given a graph $G(N,A)$, with $N = \{0, 1, 2, \ldots, n\}$ and $A$ being the set of arcs. In the general case, $G$ is complete. Arc traversal costs are known and equal to $c_{ij}$ and arc traversal times are also known and equal to $t_{ij}$ ($i \in N, j \in N$)-neither is necessarily symmetric. We assume that $c_{0i} = 0$ and $t_{ij} = 0$. Also we are given an origin/destination (O/D) matrix $d_{ij}$, representing the volume of cargo that has to go from $i$ to $j$ ($i \in N, 0, j \in N, 0 \neq j$). This matrix is not necessarily symmetric either. Each cargo is considered a distinct commodity and cannot be split. There can be as many as $n(n-1)$ such distinct cargoes to be transported.

A vehicle, at time 0 located at a separate node 0 (depot), has to visit all other nodes in $N$, pick up from each node all cargo destined to other nodes, deliver all cargo coming from other nodes (not necessarily in that order), and finally return to node 0. Each node $i \in N \setminus 0$ can be visited as many times as necessary so as to pick up and deliver cargoes originating from it and destined to it. These operations can be combined in a single stop if this is warranted. Cargoes to or from distinct nodes can co-exist on the vehicle, so long as vehicle capacity of $Q$ (a known input) is not exceeded. It is assumed that $Q \geq \max_{i \in N} d_{0i}$, otherwise the problem is infeasible. Assume finally that loading and unloading the vehicle takes negligible time. Including non-zero dwell times (which can be node-dependent and/or cargo quantity-dependent) is a straightforward extension.

In addition to vehicle trip costs, another component of the cost is the ‘delay cost’ of the cargo. This cost can be important if timely delivery of the cargo is significant. It could also be important if the time to traverse the arcs of the network and/or the quantities to be transported are non-trivial. Components of this cost may be inventory-related, such as storage, lost revenue due to delayed delivery, etc. We assume that the per unit volume and per unit time cargo delay cost is equal to $z$ for cargo waiting to be picked up (cost accrues from time 0 until cargo is on the vehicle) and to $\beta$ for cargo within the vehicle (cost accrues from time cargo is on the vehicle until cargo is delivered). Both $z$ and $\beta$ are constants, and both are $\geq 0$.

Coefficients $z$ and $\beta$ may be different for various reasons. For instance, the case $z = 0$ assumes that cargo is available at the origin in a ‘just-in-time’ fashion and related waiting or delay costs are zero. Also, these costs would generally depend on whether the cargo is at the origin’s warehouse or inside the vehicle. 2

The objective of the problem as defined above is to find a feasible route that minimizes the total costs of the trip. We shall name this problem VRPPD-G (General version of the Vehicle Routing Pickup and Delivery Problem). Other than vehicle capacity

---

1 This can be the case in long-haul problems, for instance in maritime transport.

2 If the time unit is in days, a lower bound for both $z$ and $\beta$ is $P/365$, where $P$ is the value of the cargo and $R$ the cargo owner’s cost of capital. This represents the revenue that is lost due to delayed delivery of the cargo by one day.
constraints, obviously precedence constraints also exist, in the sense that each cargo must be picked up before it is delivered.

The rest of this paper is organized as follows. Section 2 explores the relationship of this problem to other problems that have appeared in the literature. Section 3 develops a dynamic programming algorithm for the problem and tests it on some small-scale examples. Section 4 extends this to the two-vehicle case. Section 5 presents the conclusions.

2. Relation to other problems—literature

It is straightforward to see that this problem is a generalization of several pickup and delivery problems that have appeared in the literature. Perhaps the most rudimentary of these is the so-called single vehicle many-to-many advance request dial-a-ride problem (DARP), for which much has been written (see Psarasitis (1980, 1983a,b,c), Sexton and Bodin (1985a,b), Desrosiers et al. (1986), Cordeau and Laporte (2003), Cordeau et al. (2006), and Berbeglia et al. (2008), among others). The term 'many-to-many' was mainly used in the past and meant many origins and many destinations, as opposed to 'many-to-one' or 'one-to-many', where a vehicle distributed people or cargoes from or to a central depot. The DARP corresponds to the special case for which all origins are distinct, all destinations are distinct, and each origin sends return cargoes (of volume $d_n$) alone sends return cargoes (of volume $d_n$, not necessarily equal to $d_0$) to all nodes. This special case, for which $d_{ij} = 0$ except if $i = j$ are O/D and $d_0$ is zero. This corresponds (by a proper reordering of the nodes) to the case matrix $\mathbf{A}$ has non-zero entries only below the diagonal, meaning that if a cargo goes from $i$ to $j$, there is no cargo going back from $j$ to $i$. Requiring that the vehicle visit each node exactly once is tantamount for the problem to be uni-directional, since the vehicle cannot return to node $i$ with return cargo from $j$, if it previously carried cargo from $i$ to $j$.

An example of the VRPPD-U is shown in Table 4 below. The transpose of the above O/D matrix would also constitute a VRPPD-U problem.

A restricted version of VRPPD-U, also known as the multi-commodity one-to-one case, has been examined by Hernández-Pérez and Salazar-González (2009). In that problem, each node has to be visited exactly once and the restriction is because $\alpha = \beta = 0$.

<p>| Table 2 |
| VRPPD O/D matrix ($n=6$) |</p>
<table>
<thead>
<tr>
<th>ij</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<p>| Table 3 |
| VRPPD-II O/D matrix ($n=6, Z=4$) |</p>
<table>
<thead>
<tr>
<th>ij</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Author's personal copy
Finally, other special cases of our problem can be considered if some or all of the matrices \( [d_{ij}], [c_{ij}] \) or \([t_{ij}]\) are symmetric, if the latter two matrices obey the triangle inequality, and if coefficients \( \alpha \) and/or \( \beta \) are zero or equal to one another.

Having said the above about all these special cases, to our knowledge not much has been written for the general problem VRPPD-G, for which all non-diagonal O/D flows are generally non-zero and not necessarily symmetric, much less if \( \alpha \) and/or \( \beta \) are not zero and not necessarily equal to one another. It is clear that this problem is NP-hard, as a special case of it (the DARP) is NP-hard.

To be sure, VRPPD-G can be modeled as a VRPPD-U problem of 2n nodes, in which each node \( i \) is split into two copies, \( i \) and \( -i \), and whereas node \( +i \) sends flows to all nodes \( j > +i \), node \( -i \) receives flows from all nodes \( j < -i \). It can also be modeled as a VRPPD problem of \( n(n-1) \) nodes, in which each node \( i \) is split into \( (n-1) \) copies, each sending cargo to all other nodes (but not to its own copies) so that only one cargo goes from one node to another node. As such, it can be solved, at least in principle, by the spectrum of techniques used to solve the DARP or the VRPPD. However, it is not known how effective and efficient these techniques might be for problems of this structure, and surely little if anything is known as regards how the more general objective function of VRPPD-G can be taken on board. The matrix of Table 1 of the DARP is a long way from the complete matrix of the VRPPD-G, therefore one would not expect the corresponding problems to be very similar.

An illustrative example of two solutions for a 4-node VRPPD-G problem is shown in Table 5 below. Both solutions involve the same vehicle route, but two alternative pickup and delivery sequences. \( Pij \) means \( 'At node \( i \), pick up cargo destined to node \( j ' \) and \( Dij \) means \( 'At node \( j \), deliver cargo originating from node \( i ' \).

One can see in this example that node 4 is visited just once, while nodes 1, 2 and 3 twice. Also one can see that a mix of loading and unloading operations takes place at each node. In fact, the sequence can be more involved than the above, depending on vehicle capacity, as will be shown in the examples later in the paper.

It can be shown by induction that a sharp upper bound on the total number of stops of the vehicle, including its two visits to the depot, is \( n^2 + 1 \) \((n > 1)\). That corresponds to the case in which all \( d_{ij} \)'s are equal to 1, and so is \( Q \), the vehicle capacity. On the other hand, for a complete O/D matrix, a lower bound on that number is \( 2n \). Special cases may have lower upper and/or lower bounds.

### 3. A dynamic programming solution

Returning to our original problem (VRPPD-G), the algorithm that will be presented below is based on dynamic programming. It reminisces the approach of Psaraftis (1980) for the DARP, which it solves in \( O(n^2 3^n) \) time, that is, in \( O(n^2 1.73^n) \) time \((m \text{ being the number of customers} = n/2)\).

#### 3.1. State variables

To proceed, we can observe that at any step along the vehicle's route, the state of the system can be defined by the following state variables:

1) \( L \), the node representing the current location of the vehicle \((L \in \mathbb{N})\)
2) \( a \times a \) matrix \( [k_{ij}] \), where for any pair \((i,j)\) with \( i \neq j \) (both \( i \neq N(0) \)), \( k_{ij} \) is defined as follows:

\[
k_{ij} = \begin{cases} 
3 & \text{if cargo from } i \text{ to } j \text{ has not been picked up yet.} \\
2 & \text{if cargo from } i \text{ to } j \text{ is on board the vehicle,} \\
1 & \text{if cargo from } i \text{ to } j \text{ has been delivered.} 
\end{cases}
\]

By convention, if there is no cargo from \( i \) to \( j \) \((d_{ij} = 0)\), we set \( k_{ij} = 1 \) and this does not change along the vehicle's route.

When the vehicle is at node 0 before the trip starts, all \( k_{ij} = 3 \) \((d_{ij} = 0, i \neq j)\). When the trip ends, all \( k_{ij} = 1 \) \((i = j)\). In between, the \( k \)'s can take on the values of 3, 2, or 1, and they are always non-increasing as we move along the route.

Once the state is \((L,k_{ij})\), options on what to do next depend on the matrix \( [k_{ij}] \). With the exception of the case where all entries of this matrix are equal to 1 \( (\text{see paragraph 3.4 below)}, \) in all other cases the next state is \((L',k'_{ij})\), where the next matrix \( [k'_{ij}] \) is such that all of its entries are equal to those of matrix \( [k_{ij}] \). except one, that of pair \((x,y)\) \((x \neq y)\), for which \( k'_{xy} = k_{xy} - 1 \). The next node \( L' \) is a function of \( L, x \) and \( y \).

We distinguish the following cases (paragraphs 3.2 to 3.4 below).

#### 3.2. Loading

This corresponds to the case that one of the \( k \)'s, \( k_{xy} \), changes from 3 to 2. This means that loading takes place at node \( x \), which may or may not be the same as \( L \). A necessary condition for loading to take place at node \( x \) is that there be at least one \( i \) for which \( k_{ij} = 3 \). This condition may not be sufficient, as feasibility should be checked (see later). Provided feasibility is not violated, then a cargo of destination \( y \) can be loaded onto the vehicle \((y \in P = \{i : k_{ij} = 3\})\).

If set \( P \) is not empty and a loading action is taken, then the next state is \((L',k'_{ij})\), where \( L' = x \) and for all pairs \((ij)\) with \( i \neq j \), it is:

\[
k'_{ij} = \begin{cases} 
3 & \text{if } i = x \text{ and } j = y, \\
2 & \text{if cargo has been delivered,} \\
1 & \text{otherwise.} 
\end{cases}
\]

Note that even if capacity constraints are not violated, if \( \alpha < \beta \) it may make sense not to load a cargo \((x,y)\) onto the vehicle when it visits node \( x \), as this may be more expensive than if this is done later. However if \( \alpha \approx \beta \) and there are no capacity constraints, once the vehicle visits node \( x \), all cargoes originating from that node should be picked up.

---

In a sense, it also relates to (and is an extension of) the dynamic programming approach to solving the TSP, see Held and Karp (1962). Such an approach solves a TSP of \( n \) nodes in \( O(n^2 2^n) \) time.

### Table 4

<table>
<thead>
<tr>
<th>( ij )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table 5

<table>
<thead>
<tr>
<th>Node</th>
<th>Sequence 1</th>
<th>Sequence 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>1</td>
<td>P12, P13, P14</td>
<td>P14</td>
</tr>
<tr>
<td>2</td>
<td>D12, P21, P23, P24</td>
<td>P21, P23, P24</td>
</tr>
<tr>
<td>3</td>
<td>D13, D23, P31, P32</td>
<td>D23, P34, P31, P32</td>
</tr>
<tr>
<td>4</td>
<td>D14, D24, D34, P41, P42, P43</td>
<td>D14, D24, D34, P41, P42, P43</td>
</tr>
<tr>
<td>1</td>
<td>D21, D31, D41</td>
<td>D21, D31, D41, P12, P13</td>
</tr>
<tr>
<td>3</td>
<td>D43</td>
<td>D13, D43</td>
</tr>
<tr>
<td>2</td>
<td>D32, D42</td>
<td>D12, D32, D42</td>
</tr>
<tr>
<td>0</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
It can be seen that if x ≠ L, this action also involves a movement of the vehicle from L to x prior to loading at x, whereas if x = L, the vehicle only loads at L.

3.3. Unloading

This corresponds to the case that one of the k’s, kxy, changes from 2 to 1. This means that unloading takes place at node y, which may or may not be the same as L. A necessary condition for unloading to take place at y is that there be at least one i for which kxy = 2. Name the origin of the cargo to be unloaded as x ∈ U = {i : kxy = 2}.

If set U is not empty and an unloading action is taken, then the next state is \( L' = [k'] \), where L’ = y and for all pairs (i,j) with i ≠ j, it is:

\[
k'_{ij} = \begin{cases} 
k_{ij} - 1 & \text{if } i = x \text{ and } j = y, \\
0 & \text{otherwise}.
\end{cases}
\]

Obviously if set U is empty, there is no unloading option available.

It can be seen that if y ≠ L, this action also involves a movement of the vehicle from L to y prior to unloading at y, whereas if y = L the vehicle only unloads at L. It can also be seen that once a vehicle is at a node L and has several cargoes to unload there, it would not make sense to unload all of these cargoes. This observation can be exploited to reduce computational time.

3.4. return to depot

If all k’s are equal to 1, then there is no other option but to return to the depot, node 0.

3.5. Stage variable

Let and \([k] \) be stage variables, the stage variable s can be defined in terms of \( L \) and \([k] \) as follows.

Vehicle is at depot (start), \( s = 0 \)
Vehicle is at depot (end), \( s = 2n(n - 1) + 1 \)
Vehicle is at any intermediate point, \( s = 3n(n - 1) - \sum \alpha \beta \gamma \)

As s can always be defined once L and \([k] \) are known, we shall not use it explicitly in our analysis.

3.6. Incremental trip costs

The incremental trip cost associated with going from L to L’ is 

\( c_{L} \) if \( L = L' \)

But there are also other cost components. In fact, the time to go from L to L’ is \( t_{L} \) if \( L = L' \), and during that time delay costs accrue. We compute them as follows.

The total volume of cargo that is not on the vehicle and still waits to be picked up when the vehicle travels from L to L’ is \( \sum \alpha \beta \gamma \delta \) and the delay cost of that cargo for that leg of the trip is \( 2\alpha \beta \gamma \delta \).

The total volume of cargo that is on board the vehicle when it travels from L to L’ is \( \sum \alpha \beta \gamma \delta \) and the delay cost of that cargo for that leg of the trip is \( \beta \alpha \gamma \delta \).

Thus the total incremental cost of going from L to L’ is

\[
\text{INCR}(L, L') = c_{L} + t_{L} \left\{ x \sum_{i,j,k=2} d_{ij} + \beta \sum_{i,j,k=2} d_{ij} \right\} .
\]

It is obvious that \( \text{INCR}(L, L') = 0 \) if \( L' = L \).

3.7. Feasibility

A necessary condition for feasibility is that the total volume of cargo on board the vehicle is not more than \( Q \), or, that \( \sum \alpha \beta \gamma \delta \leq Q \). However, this is not a sufficient condition. Even if this condition is satisfied, if all next states of a particular state are infeasible, so is the state itself. In that sense, feasibility is recursive.

3.8. Optimal value function and optimality recursion

We are now in a position to define our optimal value function as follows.

\( V(L, [k]) = \text{Minimum possible total cost to complete the trip from node L to node 0, by executing all pending actions on pickup and delivery of the cargoes and observing capacity constraints, given that the current status of the cargoes is described by matrix \([k] \) }.

V is set to infinity (or to a very large value \( M \)) if the state \((L, [k]) \) is infeasible.

Based on all of the above, the recursive relationship for \( V \) is established as follows.

Define set \( R = \{(i,j) : i ≠ j, k_{ij} ≠ 1 \} \)

Define \( M = \text{large number} \)

If \( R = \emptyset \), then

\[
V(L, [k]) = \begin{cases} 
M & \text{if } \sum_{(i,j)\in R} d_{ij} > Q, \\
\min_{(x,y)\in R} \left\{ \text{INCR}(L, L') + V(L', [k']) \right\} & \text{otherwise},
\end{cases}
\]

where for all pairs (i,j) with i ≠ j, it is:

\[
k'_{ij} = \begin{cases} 
k_{ij} - 1 & \text{if } i = x \text{ and } j = y, \\
k_{ij} & \text{otherwise}.
\end{cases}
\]

\[
\text{INCR}(L, L') = c_{L'} + t_{L'} \left\{ x \sum_{i,j,k=2} d_{ij} + \beta \sum_{i,j,k=2} d_{ij} \right\} \]

(2)

(obviously, if \( L' = L \), \( \text{INCR}(L, L') = 0 \).

The optimal value of the problem is \( V(0, [k]) \) where \([k] \) is the ‘startup’ \([k] \) matrix, having all non-diagonal k’s equal to 3 (except those for which the corresponding entry for O/D flow d is zero, which have k equal to 1).

The DP recursion implied by (1) is a generalization of similar recursions for the DARP (Psaraftis, 1980; Psaraftis, 1983a) and for the TSP (Held and Karp, 1962). In it, the sub-problem that is defined at each particular state is recursively linked with the sub-problems that are defined for all possible ‘descendant’ states, as described by set R. A descendant state has all k’s the same with the current state except one, the one corresponding to the best next action (x,y), which is reduced by 1. In that sense, time-wise the k matrix evolves from all elements being equal to 3 (route start) to all elements being equal to 1 (route end). Computationally, the recursion is executed backwards.4

We also observe that when visiting a node, it makes no sense to load cargoes at that node before all cargoes destined to that node have been delivered, even though this could be feasible. This observation can be (and has been) exploited to reduce computational effort.

---

4 There are several alternate ways this backward recursion can be implemented, depending on the order in which states \((L, [k]) \) are looked at. In all cases, when a particular state is being worked on, all of its descendant states should have been evaluated at a prior step of the procedure.
Being able to solve the general problem VRPPD-G, this algorithm can obviously also solve all of its special cases, such as the VRPPD, the VRPPD-II and the VRPPD-U, even in their generalized forms \((x\) and/or \(\beta\) non-zero).

A straightforward (and quite useful) extension of the above algorithm is if vehicle trip costs are weighted by a user-defined "weight" \(W > 0\). If so, \(W\) should multiply \(c_{ij}\) in (2) and \(c_{ib}\) in the boundary condition \(V(L, k_i) = c_{ib}\). Introducing the weight \(W\) allows, if so desired, for vehicle trip costs to be weighted differently from delay costs, including the extreme case \(W = 0\) (objective is to minimize delay costs only). In the runs that we have implemented, we have used \(W = 0\) or 1.

3.10. Some illustrative examples

Programming the above algorithm is straightforward. We have carried out such an implementation, in order to test it for small problem sizes and make a preliminary exploration of its behavior.

Table 6 displays the optimal solution for a variety of scenarios, which include variations in:

| a | The objective function, namely coefficients \(x\) and \(\beta\), as well as weight \(W\) (as per section 3.7). |

\[\text{(a) The objective function, namely coefficients } x \text{ and } \beta \text{, as well as weight } W \text{ (as per section 3.7).} \]

\[5 \text{ We cite here the runs conducted by Hernández-Pérez and Salazar-González (2000) for a VRPPD-U type of problem, for which the O/D matrices examined were sparse: even though the maximum problem size was } n = 47, \text{ the maximum } r \text{ examined was 15 (the approach used was MILP decomposition).} \]

\[6 \text{ The computer code has been written in Fortran 95 and implemented on a PC.} \]
4. Extension to the 2-vehicle case

4.1. Solution properties

Why would anyone bother examining the 2-vehicle case? This is the case if two vehicles are available and we want to investigate how the allocation of cargoes to vehicles may be split so as to achieve the same objective as that in the single vehicle case. The fact that for each cargo that goes from $i$ to $j$ there may be a cargo going in the opposite direction might seem to fit the 2-vehicle case, because one may conceivably consider scenarios where no cargoes that travel in opposite directions between nodes pairs are carried by the same vehicle. We assume there is no transshipment, that is, if a cargo is loaded onto a vehicle, it will have to be delivered by the same vehicle.\(^7\)

---

\(^7\) Solutions where transshipment is allowed will generally achieve a lower total cost than those for which transshipment is prohibited.

---

The 2-vehicle case is obviously a special case of the more general $m$-vehicle case ($m>1$), which will not be examined here. The reason we examine the 2-vehicle case is because, as will be seen, solving it is a straightforward extension of the single vehicle case, and involves additional computational effort whose order of magnitude is no more than that of the single vehicle case.

Solving the 2-vehicle case in the classical Vehicle Routing Problem and in many of its variants involves finding an optimal partition among nodes of the problem into two disjoint sets, with each of the vehicles visiting only the nodes of each of these sets. In the VRPPD-G we are talking not about an optimal partition of nodes, but about an optimal partition of cargoes, each to be served by one of the two vehicles.\(^8\) And since each of the cargoes corresponds to a unique element of the O/D matrix, we will be looking for an optimal partition of the O/D matrix. The objective function is assumed to be the same as in the single vehicle case, minimize total cost.

The example in Table 9 shows a possible partition of a certain O/D matrix, with all cargoes served by vehicle 1 being marked by an asterisk whereas all other cargoes are served by vehicle 2.

If there are $r$ distinct non-zero cargoes, the number of possible partitions is $2^{r-1}$, including the null partition, the one that corresponds to all cargoes going on one vehicle. Note that $r$ can be as high as $n(n-1)$, depending on the variant to be examined.

A special type of partition is that for all cargoes, if cargo $(i,j)$ is on a certain vehicle, cargo $(j,i)$ is on the other vehicle. This, after a suitable reordering of the nodes, corresponds to a case like the one shown in Table 10 below. We shall call such a ‘uni-directional’ partition the $U$-partition. In it, no transpose cargo pairs are served by the same vehicle.

---

\(^8\) A node partition in this problem does not make sense. In partition ((1,2,3),(4,5,6)) there may exist cargoes that go from node 1 to node 4, from node 5 to node 3, and so on.
The number of possible U-partitions is $2^{r/2}$, or $1.41^r$, still an exponential number, which is however growing much slower than $2^r$. In a U-partition, the problem decomposes into two single vehicle VRPPD-U problems, as defined earlier, with each vehicle allocated to the corresponding VRPPD-U problem.

It is important to realize that restricting the search among possible partitions only to U-partitions is generally sub-optimal. This is shown by a simple example ($n = 3$, uncapacitated case, $\alpha = \beta = 1$).

Assume the time and cost matrices of Table 11 where $f$ and $g$ are positive constants.

Assume finally that $d_{ij} = 1$ for all $i$ and $j$ between 1 and 3 ($i \neq j$). If one is restricted only to U-partitions, it can be seen that the U-partition of Table 12 is optimal, where the corresponding routes and loading/unloading sequences are:

Vehicle 1: 0-1-2-3-0: depot, P12, P13, D12, P23, D13, D23, depot
Vehicle 2: 0-3-2-1-0: depot, P31, P32, D32, P21, D31, D21, depot

The values of the objective function are as follows:

Vehicle 1:
Trip cost: 4$f + g$
Delay cost: 8$g$($= 2g + 3g + 3g$)
Total cost: $12g$
Total cost for 2 vehicles: $12(f + g)$

If however partition is as per Table 13, that is, vehicle 1 now also serves cargo from 2 to 1, in addition to the cargo that goes from 1 to 2 (and everything else being equal), then the corresponding routes and loading/unloading sequences are:

Vehicle 1: 0-1-2-3-1-0: depot, P12, P13, D12, P23, D13, D23, depot
Vehicle 2: 0-3-2-1-0: depot, P31, P32, D32, D31, depot

Then the values of the objective function will be as follows.

Vehicle 1:
Trip cost: 4$f + g$
Delay cost: 12$g$($= 2f + 3f + 3f + 4f$)
Total cost: $16f + g$
Vehicle 2:
Trip cost: 4$g$
Delay cost: 5$g$($= 2g + 3g$)
Total cost: $9g$
Total cost for 2 vehicles: $16f + 10g$

It can be seen that if $12(f + g) > 16f + 10g$, or if $f < 0.5g$, the latter partition achieves a lower total cost than the former. See for instance the case $f = 9, g = 20$.

Therefore, restricting ourselves to examining only U-partitions is sub-optimal, and a U-partition strategy is generally a heuristic one. But as will be seen below, it makes no sense to look at heuristic methods to solve the partitioning problem.

### 4.2. Solution approach

How can the general 2-vehicle case be solved optimally? We assume that the two vehicles are identical, including their capacity $Q$.

It is clear that for a specific given partition $(S_1, S_2)$, the total cost of the optimal solution corresponding to this partition is

$$U(S_1, S_2) = V(0, [k_{ij}^{o1}]) + V(0, [k_{ij}^{o2}]),$$

where $V$ is the optimal value function of the single-vehicle case and $[k_{ij}^{o1}]$ and $[k_{ij}^{o2}]$ are the individual ‘startup’ $k$-matrices associated with the above partition.

For a given partition $(S_1, S_2)$, these ‘startup’ $k$-matrices are defined as follows for all pairs $(i,j)$ with $i \neq j$, $d(i,j) = 0$:

(a) If $(i,j) \in S_1$, then $k_{ij}^{o1} = 3$, otherwise $k_{ij}^{o1} = 1$.
(b) $k_{ij}^{o2} = 4 - k_{ij}^{o1}$.

It is indeed straightforward to observe that all (non-diagonal) elements of any ‘startup’ $k$-matrix are either 3 or 1 (not 2), and in fact are linked with one another via the following equation:

$$k_{ij}^{o1} + k_{ij}^{o2} = 4 \text{ for all pairs } (i,j), \quad i \neq j, \quad d(i,j) = 0.$$

This means that once the startup matrix of the 1st vehicle is known, the startup matrix of the 2nd vehicle is also known. If one of the elements $k_{ij}^{o1}$ in one of the startup matrices is equal to 3, the corre-

---

Table 10

<table>
<thead>
<tr>
<th>ij</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>-</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>3</td>
<td>-</td>
<td>3</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>3</td>
<td>2</td>
<td>-</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>-</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>-</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 11

<table>
<thead>
<tr>
<th>ij</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>f</td>
<td>100g</td>
<td>g</td>
</tr>
<tr>
<td>1</td>
<td>g</td>
<td>-</td>
<td>f</td>
<td>100g</td>
</tr>
<tr>
<td>2</td>
<td>100g</td>
<td>g</td>
<td>-</td>
<td>f</td>
</tr>
<tr>
<td>3</td>
<td>f</td>
<td>-</td>
<td>g</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 12

Optimal U-partition. Cargoes served by vehicle 1 are marked by an asterisk, whereas all other cargoes are served by vehicle 2.

<table>
<thead>
<tr>
<th>ij</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>1*</td>
<td>1*</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>-</td>
<td>1*</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 13

Alternate partition. Cargoes served by vehicle 1 are marked by an asterisk, whereas all other cargoes are served by vehicle 2.

<table>
<thead>
<tr>
<th>ij</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>1*</td>
<td>1*</td>
</tr>
<tr>
<td>2</td>
<td>1*</td>
<td>-</td>
<td>1*</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>
There is no way the 2-vehicle algorithm is solved. The single vehicle problem is solved only once, and the single vehicle recursion can be equal to 2, as this would mean that the corresponding cargo would be on board the vehicle when the vehicle is at node 0.

Once the single vehicle problem has been solved, the values of \( V(0,[k_{ij}^{01}]) \) and \( V(0,[k_{ij}^{02}]) \) are already available for any partition \( (S_1,S_2) \), and so are the corresponding optimal routes. These values are available since they have been computed in the course of execution of the single vehicle recursion.

To find the optimal partition \( (S_1,S_2) \) we will need to minimize \( U(S_1,S_2) \) over all possible partitions \( (S_1,S_2) \). Doing this by complete enumeration will take \( O(2^r) \) time, over and above the time spent to solving the single vehicle case.

Note that the algorithm for the 2-vehicle case is a post-optimization step that follows after the single vehicle problem has been solved. The single vehicle problem is solved only once, and the post-optimization step is executed also only once.

Due to the above sequential nature, the overall running time of the 2-vehicle algorithm is \( O(r^23^2 + 2^r) \), or still \( O(r^23^2) \), as the second step of the algorithm (picking the best partition) is computationally dominated by the first (solving the single vehicle case).

By same token, one can also extend this approach to examine a 3-partition exact scheme for the 3-vehicle case, but not a 4-partition scheme, as the former would still be possible in \( O(r^23^3) \) time, whereas the latter would involve time \( O(4^r) \).

### 4.3. Some illustrative examples

As with the single vehicle case, we have also programmed the 2-vehicle algorithm and have implemented it on a set of rudimentary problems, so as to explore the nature of the solutions. Table 14 depicts some runs, the problem instance being the same as that of section 3.9.

In Table 14, V1 and V2 pertain to results for the 1st and 2nd vehicle respectively.

One can observe, among other things, that in case 1 (which minimizes total vehicle trip costs) only one vehicle is used, and that in case 2 the optimal partition is a U-partition. No other clearly identifiable patterns can be observed from this sample, which might obviously be available by further testing of the algorithm.

### 5. Conclusions

We have developed a dynamic programming algorithm for solving a multi-commodity, capacitated pickup and delivery problem. Cargo flows are given by an origin/destination matrix which is complete and not necessarily symmetric in the general case. This problem is a generalization of several known pickup and delivery problems, as regards both problem structure and objective function. In addition to vehicle trip costs, a component of the cost function is the 'delay cost' of the cargo. This cost can be important if the value of the cargo is significant and/or if timely delivery of the cargo is significant. It could also be important if the time to traverse the arcs of the network and/or the quantities to be transported are significant. Components of this cost may be inventory-related, such as storage, lost revenue due to delayed delivery, etc.

Solution approaches were developed for the single-vehicle and two-vehicle cases. For the two-vehicle case, solutions where no cargoes that travel in opposite directions between node pairs are carried by the same vehicle were shown to generally be sub-optimal. As expected, the computational effort of the single vehicle algorithm is exponential in the number of cargoes. For the two-vehicle case, said effort is of an order of magnitude that is not higher than that of the single-vehicle case.

Obviously the contribution of this paper is mostly methodological, as it proposes an exact method for a problem on which, to this author's knowledge, not much has been reported, since all known solutions to date refer to specific special cases of it. As expected, the computational effort of such an approach is exponential and this is expected to limit maximum solvable problem size. However, lower computational times can be achieved in special cases, for instance in sparse graphs or for low values of the vehicle capacity.

Further work may involve refining this set of algorithms so as to enhance their computational efficiency and practical usefulness for realistic problem instances.

### Acknowledgments

The author thanks the Editor, two anonymous referees and Gilbert Laporte for their comments on earlier versions of the paper. I also want to thank PhD student Christos Kontovas for helping me program the algorithms of the paper.

### References


