Volume 215, Issue 3, 16 December 2011
ISSN 0377-2217
ELSEVIER
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
Available online at www.sciencedirect.com
SciVerse ScienceDirect
LAST ISSUE OF THIS VOLUME
www.elsevier.com/locate/ejor

European Journal of Operational Research 215 (2011) 572–580

Contents lists available at ScienceDirect

European Journal of Operational Research

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

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

ARTICLE INFO

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

Keywords:
Routing
Scheduling
Pickup and delivery

ABSTRACT

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.

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 $c_{ii} = 0$ and $t_{ii} = 0$. 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 ($i \in N\backslash 0$, $j \in N\backslash 0$, $i \neq j$). 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 $i \in N\backslash 0$ 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 $Q \geqslant \max_{(i,j)} d_{ij}$, 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.[1] 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 α for cargo waiting to be picked up (cost accrues from time 0 until cargo is on the vehicle) and to β for cargo within the vehicle (cost accrues from time cargo is on the vehicle until cargo is delivered). Both α and β are constants, and both are $\geqslant 0$.

Coefficients α and β may be different for various reasons. For instance, the case $\alpha = 0$ 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

* Fax: +30 2107721408.
E-mail address: hnpsar@mail.ntua.gr

[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 α and β is $PR/365$, 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.

0377-2217/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2011.06.038

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 Psaraftis (1980, 1983a,b,c), Sexton and Bodin (1985a,b), Desrosiers et al. (1986), Cordeau and Laporte (2003), Cordeau (2006), Cordeau 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 n is an even number, equal to $2m$, where m is the number of individual customers requesting service. For the DARP the O/D matrix is as follows: $d_{ij} = 1$ only for pairs of the form $j = i + m$, $1 \leqslant i \leqslant m$, and $d_{ij} = 0$ for all other pairs. An obvious example is shown in Table 1 below.

A variety of objective functions have been examined for the DARP, most common of which is the minimization of total distance traveled, or total route cost, which is equivalent to setting $\alpha = \beta = 0$ in our case. However, Psaraftis (1980) and Sexton and Bodin (1985a,b) also examined more general objective functions, some of which (addressing customer dissatisfaction due to waiting for the vehicle and spending time in the vehicle) are equivalent in substance to the one defined for VRPPD-G.

One can see that since the set of DARP nodes is split into a set of pickup-only nodes ($i = 1$ to m) and a set of delivery-only nodes ($i = m + 1$ to n), there is no sense to visit the same node more than once. Even though one may conceivably consider a scenario in which many customers originate from the same node, or even there are customers moving in opposite directions (one from node A to node B and another from B to A), the archetypal version of the DARP considers distinct origins, distinct destinations, and one person for each O/D pair and only in one direction. Under such assumptions, the typical DARP route visits each node *exactly once*, either to pick up or to deliver a customer, and always observing the ‘pickup precedes delivery’ constraints, and the vehicle capacity constraints, if any. The DARP itself is a generalization of the Traveling Salesman Problem (TSP), and, as such, is NP-hard.

A *generalization* of the DARP, which is still a special case of our problem, is the single vehicle version of the so-called *vehicle routing problem with pickup and delivery* (VRPPD) – see Kalantari et al. (1985), Desrosiers et al. (1986), Ruland and Rodin (1997), Cordeau et al. (2008), and Berbeglia et al. (2007). This has the same O/D matrix structure as the DARP, the only difference being that the non-zero elements of the O/D matrix do not take on the value of 1, but can take on any positive value, equal to the volume of cargo that goes from node i to node $i + m$. An example is shown in Table 2 below.

The usual objective function of the VRPPD, at least as it has been examined in the literature, is to minimize total distance traveled, which again corresponds to the case $\alpha = \beta = 0$ in our problem.

It is interesting to note that a significant portion of the O/D matrices in both Tables 1 and 2 have zero elements, including m entire rows and m entire columns. In fact, only $m = n/2$ elements of the matrix are non-zero.

Yet another special case, which is a variant of the VRPPD, is if all cargoes that originate from nodes $i \in N\backslash\{0,Z\}$ go to a unique destination $Z \in N\backslash 0$ (cargo volume d_{iZ}) and that node Z (a cargo collection depot) alone sends return cargoes (of volume d_{Zi}, not necessarily equal to d_{iZ}) to all nodes. This special case, for which $d_{ij} = 0$ except if i or j are Z, or the O/D matrix has only one non-zero row and one non-zero column, the Zth in both cases. This problem has been examined by Gribkovskaia et al. (2007) and is named SVRPPD (S for single vehicle). For purposes of notational consistency, we define this problem here as VRPPD-II. Table 3 below shows an example of an O/D matrix.

Here too we see a significant portion of the O/D matrix having zero elements, and here too the typical objective function is to minimize total distance traveled, which again is equivalent to putting $\alpha = \beta = 0$ in our problem.

Still another special case, which is a generalization of the VRPPD, but for which not much is known to this author, is what we can name the ‘uni-directional case’ (VRPPD-U). This is if for each pair (i,j), at least one of d_{ij} and d_{ji} is zero. This corresponds (by a proper reordering of the nodes) to the case matrix $[d_{ij}]$ 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 $\alpha = \beta = 0$.

Table 1
DARP O/D matrix ($n = 6$).

i/j	1	2	3	4	5	6
1	–	0	0	1	0	0
2	0	–	0	0	1	0
3	0	0	–	0	0	1
4	0	0	0	–	0	0
5	0	0	0	0	–	0
6	0	0	0	0	0	–

Table 2
VRPPD O/D matrix ($n = 6$).

i/j	1	2	3	4	5	6
1	–	0	0	5	0	0
2	0	–	0	0	10	0
3	0	0	–	0	0	7
4	0	0	0	–	0	0
5	0	0	0	0	–	0
6	0	0	0	0	0	–

Table 3
VRPPD-II O/D matrix ($n = 6$, $Z = 4$).

i/j	1	2	3	4	5	6
1	–	0	0	5	0	0
2	0	–	0	4	0	0
3	0	0	–	7	0	0
4	8	3	2	–	6	5
5	0	0	0	2	–	0
6	0	0	0	9	0	–

Table 4
VRPPD-U O/D matrix ($n = 6$).

i/j	1	2	3	4	5	6
1	–	0	0	0	0	0
2	5	–	0	0	0	0
3	3	7	–	0	0	0
4	8	3	2	–	0	0
5	2	4	4	2	–	0
6	4	3	5	7	2	–

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 α and/or β 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 α and/or β 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.

Table 5
Example of a 4-node problem.

Node i	Sequence 1	Sequence 2
0	–	–
1	P12, P13, P14	P14
2	D12, P21, P23, P24	P21, P23, P24
3	D13, D23, P34, P31, P32	D23, P34, P31, P32
4	D14, D24, D34, P41, P42, P43	D14, D24, D34, P41, P42, P43
1	D21, D31, D41	D21, D31, D41, P12, P13
3	D43	D13, D43
2	D32, D42	D12, D32, D42
0	–	–

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 $O(m^2 3^m)$ time, that is, in $O(n^2 1.73^n)$ time (m being the number of customers = $n/2$).[3]

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 ($L \in N$)
2) a $n \times n$ matrix $[k_{ij}]$, where for any pair (i,j) with $i \neq j$ (both $\in N \backslash 0$), 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 $k_{ij} = 3$ (for $d_{ij} \neq 0$, $i \neq j$). When the trip ends, all $k_{ij} = 1$ ($i \neq j$). 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 (see paragraph 3.4 below), in all other cases the next state is $\left(L', \left[k'_{ij}\right]\right)$, where the next matrix $\left[k'_{ij}\right]$ 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 $k_{xi} = 3$. 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 ($y \in P = \{i : k_{xi} = 3\}$).

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

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

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 $\alpha \geqslant \beta$ and there are no capacity constraints, once the vehicle visits node x, all cargoes originating from that node should be picked up.

[3] 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.

It can be seen that if $x \neq L$, 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, k_{xy}, 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 $k_{iy} = 2$. Name the origin of the cargo to be unloaded as $x \in U = \{i : k_{iL} = 2\}$.

If set U is not empty and an unloading action is taken, then the next state is $\left(L', \left[k'_{ij}\right]\right)$, where $L' = y$ and for all pairs (i,j) with $i \neq j$, it is:

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

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

It can be seen that if $y \neq L$, 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 but 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

L and $[k_{ij}]$ being state variables, the *stage variable* s can be defined in terms of L and $[k_{ij}]$ 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, $s = 3n(n-1) - \sum_{(i,j):i\neq j} k_{ij}$

As s can always be defined once L and $[k_{ij}]$ 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 $c_{LL'}$ (0 if $L' = L$). But there are also other cost components. In fact, the time to go from L to L' is $t_{LL'}$ (0 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 $\sum_{(i,j):k_{ij}=3} d_{ij}$ and the delay cost of that cargo for that leg of the trip is $\alpha t_{LL'} \sum_{(i,j):k_{ij}=3} d_{ij}$.

The total volume of cargo that is onboard the vehicle when it travels from L to L' is $\sum_{(i,j):k_{ij}=2} d_{ij}$ and the delay cost of that cargo for that leg of the trip is $\beta t_{LL'} \sum_{(i,j):k_{ij}=2} d_{ij}$.

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

$$\text{INCR}(L, L') = c_{LL'} + t_{LL'} \left\{ \alpha \sum_{(i,j):kij=3} d_{ij} + \beta \sum_{(i,j):k_{ij}=2} d_{ij} \right\}.$$

It is obvious that 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 $\sum_{(i,j):k_{ij}=2} d_{ij} \leqslant Q$. 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.

$V(L,[k_{ij}])$ = 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 $[k_{ij}]$.

V is set to infinity (or to a very large value M) if the state $(L,[k_{ij}])$ is infeasible.

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

Define set $R = \{(i,j) : i \neq j,\ k_{ij} \neq 1\}$
Define M = large number
If $R = \emptyset$, $V(L,[k_{ij}]) = c_{L0}$ (boundary condition)
If $R \neq \emptyset$, then

$$V(L, [k_{ij}]) = \begin{cases} M \text{ if } \sum\limits_{(i,j):k_{ij}=2} d_{ij} > Q, \\ \min\limits_{(x,y)\in R} \left\{ \text{INCR}(L, L') + V\left(L', \left[k'_{ij}\right]\right) \right\} \text{ otherwise,} \end{cases} \tag{1}$$

where for all pairs (i,j) with $i \neq j$, it is:

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

$$L' = \begin{cases} x \text{ if } k_{xy} = 3, \\ y \text{ if } k_{xy} = 2 \end{cases}$$

and

$$\text{INCR}(L, L') = c_{LL'} + t_{LL'} \left\{ \alpha \sum_{(i,j):k_{ij}=3} d_{ij} + \beta \sum_{(i,j):k_{ij}=2} d_{ij} \right\} \tag{2}$$

(obviously, if $L' = L$, INCR $(L,L') = 0$)

The optimal value of the problem is $V(0,[k_{ij}]^0)$ where $[k_{ij}]^0$ 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 $(L,[k_{ij}])$ 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 (α and/or β non-zero).

A straightforward (and quite useful) extension of the above algorithm is if vehicle trip costs are weighted by a user-defined "weight" $W \geqslant 0$. If so, W should multiply $c_{LL'}$ in (2) and c_{L0} in the boundary condition $V(L,[k_{ij}]) = c_{L0}$. 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.

3.9. Computational effort

The memory requirements and running time of this algorithm can be computed in a straightforward way.

Regarding memory, L is $O(n)$, and the number of possible combinations of values of the $[k]$ matrix is $O(3^r)$, where r is the number of non-zero O/D pairs, hence memory grows as $O(n3^r)$. For a complete graph, $r = n(n-1)$. For the VRPPD-U case (as defined earlier), $r = n(n-1)/2$. These values can be interpreted as upper bounds, as r will be lower if the O/D matrix is sparse.[5] For a DARP or VRPPD type of problem (see also earlier), $r = n/2$, and for a VRPPD-II type of problem (see again earlier), $r = 2(n-1)$.

Regarding computational effort, finding the minimum in (1) takes $O(r)$ time. Updating matrix $[k]$ also takes $O(n)$ time, and each of the summations in Steps 3 and 4 takes $O(r)$ time. As the sums in these steps are independent of (x,y), each iteration of the recursion takes $O(r)$ time, bringing the total computational effort to $O(r^2 3^r)$. This can be as high as $O(n^4 3^{n^2})$ in the most general case. An exception is if both α and β are zero, in which case there are no summations to be taken. In this case the computational effort reduces to $O(n^2 3^r)$.

Obviously such effort is on the high side for anything but small values of r, especially if matrix $[d]$ is complete. Lower computational times can be achieved in special cases, for instance in sparse graphs or for low values of Q. But all these times will still be exponential.

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 for selected cases.[6] Whereas no 'heavy duty' or large-scale version of the algorithm is currently available, its development is considered straightforward and is planned for the immediate future.

As an illustrative example, consider that $N = \{0,1,2,3\}$ $(n = 3, r = 6)$ and that the time and cost (non-symmetric) matrices are given by Table 6:

Table 6
Time and cost matrices $[t_{ij}] = [c_{ij}]$.

i/j	0	1	2	3
0	–	5	7	3
1	4	–	5	7
2	8	6	–	6
3	3	8	6	–

The (non-symmetric) O/D matrix is given by Table 7.

Table 7
O/D matrix $[d_{ij}]$.

i/j	1	2	3
1	–	5	2
2	1	–	4
3	6	3	–

This means that at each node of this problem (other than the depot) there are two cargoes, each destined to each of the other two nodes. There are $r = 6$ such cargoes in total.

Table 8 displays the optimal solution for a variety of scenarios, which include variations in:

(a) The objective function, namely coefficients α and β, as well as weight W (as per section 3.7).

(b) The vehicle capacity Q.

In Table 8, weight W takes on the values of 0 or 1. Z_c is the total vehicle trip cost (irrespective of whether or not this is the objective to be optimized) and Z is the actual objective to be optimized (total cost including delay cost).

Note that a vehicle capacity of $Q = 21$ or above corresponds to the unconstrained case and that $Q = 6$ is the minimum vehicle capacity for the problem to be feasible (the bottleneck here being $d(3,1) = 6$).

Some comments on Table 8 are as follows.

1. One can generally see that the optimal route and pickup/delivery sequence change, sometimes drastically, with a change in the objective function and/or vehicle capacity. As expected, the vehicle visits nodes generally more than once.
2. Cases 1 to 4 refer to the scenario where the objective to be optimized is the total vehicle trip cost, that is, if delay costs are assumed zero ($\alpha = \beta = 0$, $W = 1$). It turns out that any capacity constraint of Q of 9 or more is essentially superfluous. Note the (expected) deterioration in the objective function optimal value when Q goes from 9 to 6.
3. Among these cases note case 4, in which $Q = 6$, whose solution is of the form 'Pij–Dij' for about half the route (vehicle loads only one cargo and then immediately delivers it). However, in segment [P21, P23, D21, P13, D13, D23] more cargoes can fit into the vehicle and this is exploited.
4. Cases 5 to 9 have $W = 0$, that is, in these cases only delay costs are important. Actually in cases 5 and 8 it is $\alpha = \beta = 1$ (delay cost in the vehicle same as delay cost outside the vehicle). Case 6 has $\alpha = 1$, $\beta = 0$ (delay costs count only for cargoes waiting to be picked up) whereas case 7 has $\alpha = 0$, $\beta = 1$ (delay costs count only while cargo is in transit).
5. Note the output of case 7. Even without capacity constraints, the solution is a sequence of 'Pij–Dij' pairs, each involving a direct shipment of each cargo from its origin to its destination. Indeed, if delay costs count only while in transit and there are no other costs, any solution that involves putting more than one cargo onto the vehicle would have higher delay costs, as some of the cargoes would travel a circuitous route.
6. Our set of runs finally includes case 10, in which $\alpha = \beta = W = 1$ and $Q = 6$. Contrast this with case 4 ($\alpha = \beta = 0$, $W = 1$ and $Q = 6$).

In terms of how the k-matrix evolves in an optimal sequence, this depends on the particular case. In case 1, the evolution is as follows.

- Vehicle initially is at depot and all k's are equal to 3.

[5] We cite here the runs conducted by Hernández-Pérez and Salazar-González (2009) for a VRPPD-U type of problem, for which the O/D matrices examined were sparse: even though the maximum problem size was $n = 47$, the maximum r examined was 15 (the approach used was MIP decomposition).

[6] The computer code has been written in Fortran 95 and implemented on a PC.

Table 8
Sample results.

Case	1	2	3	4	5	6	7	8	9	10
α	0	0	0	0	1	1	0	1	1	1
β	0	0	0	0	1	0	1	1	0	1
W	1	1	1	1	0	0	0	0	0	1
Q	9+	8	7	6	21	21	21	10	10	6
Z_c	29	36	37	43	38	29	75	40	34	44
Z	29	36	37	43	356	177	135	386	184	470
	P31	P12	P32	P31	P31	P31	P12	P31	P31	P31
	P32	P13	D32	D31	P32	P32	D12	P32	P32	D31
	D32	D13	P21	P12	D31	D32	P13	D31	D31	P12
	P21	P32	P23	D12	P12	P21	D13	P12	P12	D12
	D21	D12	D23	P21	P13	P23	P21	D12	P13	P21
Optimal pickup and delivery sequence (depot at start and end not shown)	D31	D32	P31	P23	D12	D21	D21	D32	D12	P23
	P12	P21	D21	D21	D32	D31	P23	P21	D32	D23
	P13	P23	D31	P13	P21	P12	D23	P23	P21	P32
	D12	D23	P12	D13	P23	P13	P31	D23	P23	D32
	P23	P31	P13	D23	D31	D12	D31	D21	D13	P13
	D13	D21	D12	P32	D23	D13	P32	P13	D23	D21
	D23	D31	D13	D32	D21	D23	D32	D13	D21	D13

- P31: Go to node 3 to pick up cargo to node 1; change k_{31} to 2 (all other k's the same).
- P32: Remain at node 3 to pick up cargo to node 2; change k_{32} to 2 (all other k's the same).
- D32: Go to node 2 to deliver cargo from node 3; change k_{32} to 1 (all other k's the same).
- P21: Remain at node 2 to pick up cargo to node 1; change k_{21} to 2 (all other k's the same).
- D21: Go to node 1 to deliver cargo from node 2; change k_{21} to 1 (all other k's the same).
- D31: Remain at node 1 to deliver cargo from node 3; change k_{31} to 1 (all other k's the same).
- P12: Remain node 1 to pick up cargo to node 2; change k_{12} to 2 (all other k's the same).
- P13: Remain at node 1 to pick up cargo to node 3; change k_{13} to 2 (all other k's the same).
- D12: Go to node 2 to deliver cargo from node 1; change k_{12} to 1 (all other k's the same).
- P23: Remain at node 2 to pick up cargo to node 3; change k_{23} to 2 (all other k's the same).
- D13: Go to node 3 to deliver cargo from node 1; change k_{13} to 1 (all other k's the same).
- D23: Remain at node 3 to deliver cargo from node 2; change k_{23} to 1 (all other k's the same).
- Vehicle returns to depot and all k's are equal to 1.

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 node 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]

Table 9
Example of an O/D matrix partition ($n = 6$).

i/j	1	2	3	4	5	6
1	–	7*	5*	4*	2	8
2	5	–	6	4	5*	3*
3	3*	7*	–	3	1	7
4	8	3	2	–	3*	2*
5	2	4*	4*	2*	–	6
6	4	3	5	7*	2*	–

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.

[7] Solutions where transshipment is allowed will generally achieve a lower total cost than those for which transshipment is prohibited.

[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.

Table 10
Example of an O/D matrix U-partition ($n = 6$). Cargoes marked by an asterisk are served by vehicle 1, whereas all other cargoes are served by vehicle 2.

i/j	1	2	3	4	5	6
1	–	7*	5*	4*	2*	8*
2	5	–	6*	4*	5*	3*
3	3	7	–	3*	1*	7*
4	8	3	2	–	3*	2*
5	2	4	4	2	–	6*
6	4	3	5	7	2	–

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-1}. 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: $4f$
Delay cost: $8f(=2f + 3f + 3f)$
Total cost: $12f$

Vehicle 2:
Trip cost: $4g$
Delay cost: $8g$ $(=2g + 3g + 3g)$
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, P21, P23, D13, D23, D21, 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: $4f + g$
Delay cost: $12f(=2f + 3f + 3f + 4f)$
Total cost: $16f + g$

Vehicle 2:
Trip cost: $4g$
Delay cost: $5g(=2g + 3g)$
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.

Table 11
Time and cost matrices $[t_{ij}] = [c_{ij}]$.

i/j	0	1	2	3
0	–	f	$100g$	g
1	g	–	f	$100g$
2	$100g$	g	–	f
3	f	f	g	–

Table 12
Optimal U-partition. Cargoes served by vehicle 1 are marked by an asterisk, whereas all other cargoes are served by vehicle 2.

i/j	1	2	3
1	–	1*	1*
2	1	–	1*
3	1	1	–

Table 13
Alternate partition. Cargoes served by vehicle 1 are marked by an asterisk, whereas all other cargoes are served by vehicle 2.

i/j	1	2	3
1	–	1*	1*
2	1*	–	1*
3	1	1	–

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

$$U(S_1, S_2) = V(0, [k_{ij}]^{01}) + V(0, [k_{ij}]^{02}),$$

where V is the optimal value function of the single-vehicle case and $[k_{ij}]^{01}$ and $[k_{ij}]^{02}$ 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 $i \neq j$, $d(i,j) \neq 0$:

(a) If $(i,j) \in S_1$, then $k_{ij}^{01} = 3$, otherwise $k_{ij}^{01} = 1$.
(b) $k_{ij}^{02} = 4 - k_{ij}^{01}$.

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:

$$k_{ij}^{01} + k_{ij}^{02} = 4 \text{ for all pairs} (i,j), \quad i \neq j, \quad d(i,j) \neq 0.$$

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}^{01} in one of the startup matrices is equal to 3, the corre-

Table 14
Illustrative runs, 2-vehicle case.

Case	1		2		3		4		5	
α	0		1		1		1		1	
β	0		1		1		1		0	
W	1		1		1		1		0	
Q	9+		21		8		6		6	
Vehicle	V1	V2	V1	V2	V1	V2	V1	V2	V1	V2
Z	29	0	151	165	137	215	105	258	84	73
Z_{total}	29		316		352		363		157	
Optimal pickup and delivery sequence (depot at start and end not shown)	P31	idle	P31	P12	P32	P31	P32	P31	P32	P31
	P32		P32	P13	D32	D31	D32	D31	D32	D31
	D32		D32	D12	P21	P12	P23	P12	P21	P12
	P21		P21	P23	P23	P13	D23	D12	P23	D12
	D21		D21	D13	D23	D12		P21	D21	
	D31		D31	D23	D21	D13		D21	P13	
	P12							P13	D13	
	P13							D13	D23	
	D12									
	P23									
	D13									
	D23									

sponding element k_{ij}^{02} in the other matrix is equal to 1, and vice versa. There is no way k_{ij}^{01} or k_{ij}^{02} 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 $O(r^2 3^r + 2^r)$, or still $O(r^2 3^r)$, 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 $O(r^2 3^r)$ 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

Berbeglia, Cordeau, J.-F., Gribkovskaia, I., Laporte, G., 2007. Static pickup and delivery problems: A classification scheme and survey. TOP 15, 1–31.

Cordeau, J.-F., 2006. A branch-and-cut algorithm for the dial-a-ride problem. Operations Research 54, 573–586.

Cordeau, J.-F., Laporte, G., 2003. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. Transportation Research B 37, 579–594.

Cordeau, J.-F., Laporte, G., Ropke, S., 2008. Recent models for one-to-one pickup and delivery problems. In: Golden, B.L., Raghavan, S., Wasil, E.A. (Eds.), The Vehicle Routing Problem: Latest Advances and New Challenges. Springer, New York, pp. 327–357.

Desrosiers, J., Dumas, Y., Soumis, F., 1986. A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows. American Journal of the Mathematical and Management Sciences 6, 301–325.

Gribkovskaia, I., Halskau sr, Ø., Laporte, G., Vlcek, M., 2007. General solutions to the single vehicle routing problem with pickups and deliveries. European Journal of Operational Research 180 (2007), 568–584.

Held, M., Karp, R.M., 1962. A dynamic programming approach to sequencing problems. Journal of the Society for Industrial and Applied Mathematics 10 (1), 196–210.

Hernández-Pérez, H., Salazar-González, J.-J., 2009. The multi-commodity one-to-one pickup-and-delivery traveling salesman problem. European Journal of Operational Research 196, 987–995.

Kalantari, B., Hill, A.V., Arora, S.R., 1985. An algorithm for the traveling salesman problem with pickup and delivery customers. European Journal of Operational Research 22 (1985), 377–386.

Psaraftis, H.N., 1980. A dynamic programming approach to the single vehicle, many-to-many immediate request dial-a-ride problem. Transportation Science 14, 130–154.

Psaraftis, H.N., 1983a. An exact algorithm for the single-vehicle many-tomany dial-a-ride problem with time windows. Transportation Science 17, 351–357.

Psaraftis, H.N., 1983b. k-interchange procedures for local search in a precedence-constrained routing problem. European Journal of Operational Research 13, 391–402.

Psaraftis, H.N., 1983c. Analysis of an $O(N^2)$ heuristic for the single vehicle many-to-many Euclidean dial-a-ride problem. Transportation Research B 17, 133–145.

Ruland, K.S., Rodin, E.Y., 1997. The pickup and delivery problem: Faces and branch-and-cut algorithm. Computers & Mathematics with Applications 33, 1–13.

Sexton, T.R., Bodin, L.D., 1985a. Optimizing single vehicle many-to-many operations with desired delivery times: I. Scheduling. Transportation Science 19, 378–410.

Sexton, T.R., Bodin, L.D., 1985b. Optimizing single vehicle many-to-many operations with desired delivery times: II. Routing. Transportation Science 19, 411–435.