Online Supplement to "A Branch-and-Bound Approach to the Traveling Salesman Problem with a Drone"

Stefan Poikonen[1], Bruce Golden[2], and Edward Wasil[3]

[1]Department of Mathematics, University of Maryland, College Park, MD 20742, USA - spoikone@math.umd.edu
[2]R.H. Smith School of Business, University of Maryland, College Park, MD 20742, USA - bgolden@rhsmith.umd.edu
[3]Kogod School of Business, American University, Washington, D.C. 20016, USA - ewasil@american.edu

Part A: Model Generalizations

Drone Service Times

Suppose each drone delivery requires a service time of st. This may be embedded in the model by increasing the values of $c_d(i,j), \forall i,j$ by $st/2$. Any drone operation triplet (i,j,k) considers a drone flight time of $c_d(i,k)+c_d(k,j)$. Each term has been increased by $st/2$, thus any drone operation has a total trip time that has increased by st, as desired.

Launch and Retrieval Times

Assume that there exists some overhead time delay (d_l) that is experienced by both the truck and the drone whenever the drone is launched. Similarly, suppose there exists some delay (d_r) experienced by both the truck and the drone whenever a drone is retrieved. Our model may be extended to consider both of these simply by altering the computation of $T(i,j,k)$. First, compute $T(i,j,k)$ in the ordinary way (i.e., as it appears in BAB). Then, if the drone operation (i,j,k) contains a single drone launch (i.e., $k>0$), then increase $T(i,j,k)$ by d_l+d_r. If the drone operation (i,j,k) does not contain a drone launch (i.e., $k=0$), then do not increase $T(i,j,k)$.

One way to model battery swap time is to assume that it occurs immediately when the drone lands. During this swap, assume that the truck and

drone are both stationary. Thus battery swap time can be included as part of the retrieval time d_r.

Precedence Constraints

Consider routes that are subject to precedence constraints. For example, suppose we require that if package locations P_1, P_3, P_5 are each visited by the same vehicle, then P_3 is visited first, followed by P_5, then P_1. We enforce this by fixing the sequence $[0, 3, 5, 1, 0]$ to the root node in our branch-and-bound tree. These precedence constraints improve computational speed.

Part B: Pseudocode describing BAB

Main Function:
%Initialization%
$RootNode.Sequence = [0, 1, 2, 0]$
$RootNode.LB = ep(RootNode.Sequence)$
$RootNode.UB = \infty$
$Nodes = \{RootNode\}$
$UnexploredNodes = Nodes$

While($RootNode.LB/TER < RootNode.UB$)
    %Select the node in our tree with the smallest lower bound.%
    $CurrentNode = argmin_{node \in UnexploredNodes}(node.LB)$
    $Children = MakeChildren(CurrentNode)$
    For each *node* in $Children$
        $node.LB = ep(node.Sequence)$
        If *node.Sequence* contains all package locations, then:
            $node.UB = ep(node.Sequence)$
            $node.LB =$ `INF`
        Else:
            $node.UB =$ `INF`
    End For
    %Remove $CurrentNode$ from the set of unexplored nodes.%
    $UnexploredNodes = UnexploredNodes - \{CurrentNode\}$
    %Update upper and lower bounds of all ancestors of $CurrentNode$%
    Do:
        $CurrentNode.UB = min_{child \in Children}(child.UB)$
        $CurrentNode.LB = min_{child \in Children}(child.LB)$
        $CurrentNode = CurrentNode.Parent$
        While($CurrentNode != RootNode$), Loop
End While

Other Functions and Attributes:
function $ep(S)$: This function returns the objective value of the exact partitioning function, where S is provided as the input sequence.

function $MakeChildren(node)$: Produces children in the manner described in section 4.1, paragraph 2 for *node*. Each child node is then added to the sets $Nodes$ and $UnexploredNodes$. This function returns the set of nodes created.

attributes $.LB$ and $.UB$: Each $node \in Nodes$ has these attributes that represent the upper bound and lower bound of *node*.

attribute $.Sequence$: Each $node \in Nodes$ has this attribute that represents the sequence associated with *node*.

attribute $.Parent$: Each $node \in Nodes$ has this attribute. $node.Parent$ is the parent of $node$ in the branch-and-bound tree.

Part C: Technical Details of the Function *ep*

The function *ep* is identical in every way to *aep*, except that it modifies the computation of each $T(i,j)$ to account for the case of a truck remaining stationary while a drone is in flight.

Recall that $T(i,j)$ represents the minimum time required to begin with the truck and the drone at P_i, end with the truck and the drone at P_j, and service all $P_l \in P$ such that $i < l < j$. When $k > 0$, any triplet (i,j,k) implies that P_k is serviced by the drone and all other package locations between P_i and P_j are serviced by the truck. The value $k = -1$ accounts for the case that $j = i + 1$ and no drone delivery occurs between P_i and P_j. The cases where $k = -1$ and $k > 0$ are considered by *aep*. We now introduce two new cases, $k = -2$ and $k = -3$ that consider the ability of a drone to launch and land at the same node.

Suppose we want to compute $T(i,j)$ to account for the case of a stationary truck where $j = i + x$. There are $x - 1$ packages between P_i and P_j, namely $P_{i+1}, P_{i+2}, ..., P_{j-1}$. We must consider two new scenarios.

In the first scenario, the truck remains stationary at P_i while the drone flies out and back to P_{i+1}, P_{i+2}, ..., and P_{j-1}, and then finally the truck and drone travel together to P_j. This operation is denoted by $T(i,j,-2)$ and

$$T(i,j,-2) = c_t(j-1,j) + \sum_{v=i+1}^{j-1} 2c_d(i,v).$$

The second scenario is more complicated. The truck is stationary at P_i while the drone flies out and back to P_{i+1}, P_{i+2}, ..., and P_{i+z} where $1 < z < j - 1$. After the drone returns from P_{i+z} to the stationary truck at P_i, the drone then launches to P_y (where $i + z < y < j$). While the drone is delivering to P_y, the truck delivers to each P_w such that $i + z < w < j$ and $w \neq y$. The truck and drone then rendezvous at P_j. We define

$$T(i,j,z,y) = \sum_{v=i+1}^{i+z} 2c_d(i,v) + max(c_d(i,y) + c_d(y,j), truckDriveTime),$$

where

$$truckDriveTime = \sum_{w=i+z}^{y-2} c_t(w,w+1) + \sum_{w=y+1}^{j-1} c_t(w,w+1) + c_t(y-1,y+1).$$

Next, we set $T(i,j,-3) = min_{z,y}T(i,j,z,y)$. Finally, we set $T(i,j) = min_k(T(i,j,k))$ and apply the Bellman-Ford equation.

In all cases, $k = -3, -2, -1$ and $k > 0$, $T(i,j,k)$ is the cost of beginning at P_i with the truck and the drone present, ending at P_j with the truck and the

drone present, and delivering to each intermediate customer between P_i and P_j. The cases $k = -3$ and $k = -2$ have the truck stationary during a drone delivery. When $k = -1$, there is no drone delivery in the operation and $k > 0$ implies that P_k is a package delivered by the drone in the operation. The function *aep* does not consider $k = -3$ and $k = -2$.

Part D: Integer Program to Compute C

The integer program to compute C, the maximum number of vertices a truck could possibly visit in R time units, is given below. The value of C is equal to the final objective value of the integer program.

$$\texttt{maximize}(\sum_i \sum_j x_{i,j})$$

$$\texttt{subject to:}$$

$$\sum_i \sum_j (c_t(i,j) * (x_{i,j} - y_{i,j})) \leq R \qquad \texttt{(D.1)}$$

$$\sum_i x_{i,j} = \sum_i x_{j,i}, \forall j \in P \qquad \texttt{(D.2)}$$

$$\sum_i \sum_j y_{i,j} \leq 1 \qquad \texttt{(D.3)}$$

$$x_{i,j} \geq y_{i,j}, \forall i,j \in P \qquad \texttt{(D.4)}$$

$$x_{i,j}, y_{i,j} \in \{0,1\}, \forall i,j \in P \qquad \texttt{(D.5)}$$

The idea is to model the truck path as a simple closed tour with flow constraints, where we remove a single edge. If $x_{i,j} = 1$, then the closed tour includes the driving segment from P_i to P_j. If $y_{i,j} = 1$, then we should subsequently remove the driving segment from P_i to P_j.

Constraint (D.1) allows for the total driving time to be at most R time units. Constraint (D.2) combined with (D.5) ensures a single, connected (i.e., non-disjoint), closed tour. Constraint (D.3) allows the removal of at most one driving segment from the closed tour, so that we preserve a connected open path. Constraint (D.4) implies that we cannot remove an edge that has not been included in the closed tour. Constraint (D.5) forces all edge choices to be binary. Finally, the objective counts the number of edges in the closed tour. This is equal to the number of vertices visited by the open path formed from removing a single edge from the closed tour.