Dynamic Programming Approaches for the Traveling Salesman Problem with Drone

Paul Bouman
Erasmus School of Economics, Erasmus University, The Netherlands
Niels Agatz and Marie Schmidt
Rotterdam School of Management, Erasmus University, The Netherlands
Email:bouman@ese.eur.nl

Abstract

A promising new delivery model involves the use of a delivery truck that collaborates with a drone to make deliveries. Effectively combining a drone and a truck gives rise to a new planning problem that is known as the Traveling Salesman Problem with Drone (TSP-D). This paper presents an exact solution approach for the TSP-D based on dynamic programming and present experimental results of different dynamic programming based heuristics. Our numerical experiments show that our approach can solve larger problems than the mathematical programming approaches that have been presented in the literature thus far. Moreover, we show that restrictions on the number of operations can help significantly reduce the solution times while having relatively little impact on the overall solution quality.

Keywords: Traveling salesman problem, Vehicle routing, Drones, Dynamic Programming

1 Introduction

Several Internet retailers and logistics service providers including Amazon, Singapore post and DHL are experimenting with the use of drones to support the delivery of parcels and mail. One promising new operating model is the use of a regular delivery truck that collaborates with a drone to make deliveries. This way, the high capacity and long range of the truck can be combined with the speed and flexibility of a drone.

Together with the University of Cincinnati, Amp Electric Vehicles supports this new operating model by developing a drone that deploys from a compartment in the roof of an electric delivery truck. A new Mercedes Benz concept vehicle called 'Vision Van' also includes roof-top drones (Condliffe, 2016) and UPS recently tested launching a drone from the roof of a delivery van (Steward, 2017).

Type of approach	Solution procedure
Exact methods	Integer linear programming (Agatz et al., 2017)
	Dynamic programming (this paper)
Approximation algorithms	(Agatz et al., 2017)
Heuristic methods	Simple heuristics (Murray and Chu, 2015; Ha et al., 2015)
	Genetic algorithms (Ferrandez et al., 2016)
	Simulated annealing (Ponza, 2016)

Table 1: Solution procedures proposed for the TSP-D

Effectively combining a drone and a truck gives rise to a new planning problem that we call the Traveling Salesman Problem with Drone (TSP-D). The problem aims to find a route for both the delivery vehicle and the drone that minimizes the total joint time to serve all delivery tasks. Due to its limited capacity, the drone has to return to the vehicle to pick up the parcel before each new delivery. For this reason, the route of the drone need to be synchronized with that of the truck.

The problem has only recently started to receive some attention from the transportation optimization community. Wang et al. (2016) and Poikonen et al. (2017) analyze the theoretical benefits of using one or more drones together with one or more delivery vehicles to make deliveries. Carlsson and Song (2017) use continuous approximation to derive analytical formulas to estimate the expected delivery costs in this setting. Table 1 provides an overview of the different solution approaches that have been proposed in the literature for several variants of the problem. We see that so far, most research has solely focused on heuristic approaches. Exact solution approaches based on mathematical programming so far are only capable of solving small instances of the problem, i.e. up to ten locations (Agatz et al., 2017). In this contribution, we propose an exact approach based on dynamic programming that is able to solve larger instances.

For the classic Traveling Salesman Problem (TSP) Held and Karp (1962); Bellman (1962) first proposed a dynamic programming approach. For the general TSP without additional assumptions, this is the exact algorithm with the best known worst-case running time to this day (Applegate et al., 2011). Dynamic programming approaches have been proposed for various variants of the TSP, including the single vehicle dial-a-ride problem (Psaraftis, 1980), the time-dependent TSP (Malandraki and Daskin, 1992) and the TSP with time window and precedence constraints (Mingozzi et al., 1997). Dynamic programming approaches also have been used as the basis for heuristics for various TSP and VRP problems (Malandraki and Dial, 1996; Kok et al., 2010).

In this paper, we introduce a dynamic programming approach to solve the TSP-D problem and extend this approach to develop an A^* algorithm. Furthermore, we investigate the impact of restrictions on the number of locations the truck may visit while

the drone is performing an delivery on the computation time and solution quality by an extensive computational study.

The remainder of this paper is organized as follows. In Section 2, we formally define the TSP-D. In Section 3, we present a dynamic programming algorithm for the TSP-D and also present two ways to speed up the algorithm. Section 4 presents the results of an extensive numerical study. Finally, in Section 5, we offer some final remarks and directions for future research.

2 Problem description

The TSP-D can be modeled as a set V of n locations, which include a depot v_0 and n-1 customer locations. Furthermore, two cost functions $c, c^d : V^2 \to \mathbb{R}$ model distances or travel times between the locations. We generally assume that $c(v, w)$ stands for the driving time of the truck driving from v to w, and $c^d(v, w)$ stands for the time it takes the drone to fly from v to w, but other types of non-negative cost functions can be considered as well.

The objective of the TSP-D is to find the minimum cost tour which serves all customer locations by either the truck or the drone. We assume that the drone has unit-capacity and has to pick up a new parcel at the truck after each delivery. Moreover, the pickup of parcels from the truck can only take place at the customer locations, i.e. the drone can only land on and depart from the truck while it is located at a node.

We define a constant α, that relates the speed of the drone and the speed of the truck to each other such that the drone can be at most α times faster than the truck for a pair of locations. Formally, we say that α is the minimum value for which $\alpha c^d(v, w) \geq c(v, w)$ for every pair of locations $v, w \in V$ that have non-zero drone costs $c^d(v, w)$. The special case where $\alpha c(v, w) = c^d(v, w)$ for any pair of locations can be interpreted as a situation where the drone and truck travel via the same network, but the drone is a factor α faster than the truck.

For a given tour, we distinguish between the following types of nodes:

Drone node: a node that is visited by the drone separated from the truck

Truck node: a node that is visited by the truck separated from the drone

Combined node: a node that is visited by both truck and drone

To compute the time needed for a tour, we have to consider that the two vehicles need to be synchronized, i.e., they have to wait for each other at the combined nodes. Hence, we decompose the tour into a sequence of *operations* $(o_1, o_2, \ldots, o_l)$.

An operation o_k consists of two combined nodes, called *start node* and *end node*, at most one drone node, and potentially several truck nodes. In an operation, the drone

departs from the truck at the start node, then serves the drone node and meets up with the truck again at the end node. Figure 1 provides an example of an operation (Agatz et al., 2017). The truck can travel directly from the start node to the end node or can visit any number of truck nodes in between. Moreover, the truck can also wait at the start node for the drone to return. In this case, the start node is equal to the end node. We obtain the time duration $t(o)$ of an operation o as the maximum over the truck driving time needed to drive between the nodes as described above, and the drone flying time from start node to fly node to end node.

To compute the time duration of the full tour, we simply sum up the time durations of all operations it contains.

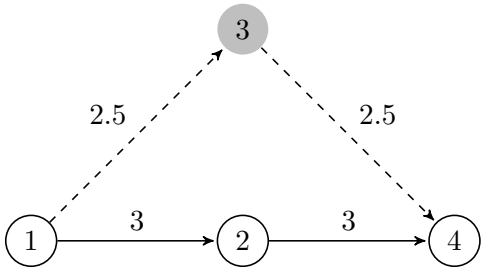

Figure 1: An operation with a combined node serving as start node (1), a combined node serving as end node (4), a drone node (3) and a truck node (2)

3 Solution approaches

3.1 Dynamic Programming approach

This sections presents a dynamic programming approach for the TSP-D based on the well-known Bellman-Held-Karp dynamic programming algorithm for the TSP (Held and Karp, 1962; Bellman, 1962). Our approach consists of the following three passes.

1. Enumerate the shortest paths for the truck for every start node, end node and set of truck nodes covered by the path.

2. Combine these truck paths with drone nodes to obtain *efficient operations*, i.e. operations where the truck nodes are covered using a shortest path along the truck nodes, for any given combination of start node, end node and set of nodes covered.

3. Compute the optimal sequence of these operations such that all locations are covered and the sequence start and ends at the depot.

Dynamic Programming can be used to find optimal solutions to problems, if the optimal solutions have an *optimal substructure*, i.e. they can be split up in an optimal solution to

a smaller sub-problem and some cost or profit that is independent from that optimal solution. In our exposition, we focus on the recursive structure of the optimal solution of the TSP-D, and provide pseudo-code for the implementation of this structure to compute the optional solutions.

The optimal solution for the regular TSP can be considered as a path that starts at an arbitrary origin point, which we assume to be v_0 without loss of generality, visits all locations in V and ends at the point where it started. Let us call $D_{\text{TSP}}(S, v)$ the sub-problem of finding the shortest path that starts at the origin point, visits all locations in S and ends at location v. The shortest path of a sub-problem can be broken down into two parts: the last arc on the path that goes from some location $w \in S$ to v, and a shortest path that starts at the origin point, visits all locations in $S \setminus \{v\}$, and ends in w. If the path for the full set S does not contain the shortest sub-path for $S \setminus \{v\}$, we can obtain a better solution by replacing this sub-path with the shortest one, contradicting the fact that the path for S was shortest. As a consequence, the sub-problem $D_{\text{TSP}}(S, v)$ can be solved by considering all arcs $(w, v) : w \in S \setminus \{v\}$, and adding each of those arcs to the solution for the smaller sub-problem $D_{\text{TSP}}(S \setminus \{v\}, w)$. This structure, in which the optimal solution of a sub-problem can be expressed in terms of smaller sub-problems can be formalized by means of a recursive formula as follows:

$$D_{\text{TSP}}(S, w) = \begin{cases} c(v_0, w) & \text{if } S = \{w\} \\ \infty & \text{if } w \notin S \\ \min\limits_{v \in S \setminus \{w\}} D_{\text{TSP}}(S \setminus \{w\}, v) + c(v, w) & \text{otherwise} \end{cases} \tag{1}$$

We can see that there are $n \cdot 2^n$ sub-problems to solve here, as S can be any subset of V. For each sub-problem in the recurrence relation we only consider at most n smaller sub-problems, and as a result this algorithm can be run in $O(n^2 \cdot 2^n)$ time. A possible implementation of an algorithm that exploits this structure in a bottom up fashion is presented in Algorithm 1.

In order to construct an optimal sequence of operations for the TSP-D, we only need to consider operations that start and end at a pair of locations and cover a set of locations with the lowest cost possible. As mentioned earlier, an operation for which the truck path is the shortest path visiting all the truck-only nodes of that operation is called an *efficient operation*. An optimal TSP-D tour can be constructed using efficient operations only as a TSP-D tour will not become longer when an inefficient operation is replaced by an efficient operation. As a consequence, we need only consider the sub-problem of finding an efficient operation associated with every triplet (S, v, w), where the operation starts at location v, ends at location w and visits all locations in S.

As the first pass of our three pass approach, we adapt the dynamic programming approach for the regular TSP to include a sub-problem of finding the shortest truck path

Algorithm 1: Dynamic Programming algorithm for the original TSP

Data: A set of locations V, an arbitrary location $v_0 \in V$ and cost function c

Result: A shortest tour that visits all locations in V

```
1  Initialize D_TSP with values ∞ ;
2  Initialize a table P to retain predecessor arcs ;
3  foreach w ∈ V do
4      D_TSP({w}, w) ← c(v_0, w) ;
5  for i = 2, ..., |V| do
6      for S ⊆ V where |S| = i do
7          foreach w ∈ S do
8              foreach u ∈ S \ {w} do
9                  v ← D_TSP(S \ {w}, u) + c(u, w) ;
10                 if v < D_TSP(S, w) then
11                     D_TSP(S, w) ← v ;
12                     P(S, w) ← (u, w) ;
13 return path obtained by backtracking over arcs in P starting at P(V, v_0) ;
```

for each of these triplets. Each of these sub-problems can be solved based on smaller sub-problems using the same idea as for the regular TSP:

$$D_{\text{T}}(S, v, w) = \begin{cases} 0 & \text{if } v = w \wedge S = \{v\} \\ c(v, w) & \text{if } v \neq w \text{ and } S = \{v, w\} \\ \max\limits_{u \in S \setminus \{v,w\}} D_{\text{T}}(S \setminus \{w\}, v, u) + c_t(u, w) & \text{otherwise} \end{cases} \tag{2}$$

The number of sub-problems we need to solve to compute table D_{T} is $2^n \cdot n^2$. By extending the approach presented in Algorithm 1, the computation of the table can be performed in $O(2^n \cdot n^3)$ time.

In the second pass, we expand the table of truck paths to create a table of operations, by adding the drone movement on top of the truck paths, if that improves the costs required to cover the same set of nodes. The problem of finding an efficient operation that starts in v, ends in w and visits all locations in S can be broken down into the problem of selecting a single drone d location from $S \setminus \{v, w\}$ and combining the flight of the drone via this location with the shortest truck path that starts in v, ends in w and visits all locations in $S \setminus \{d\}$. As the sub-problem of finding a shortest truck path was already solved when we constructed table D_{T}, the table D_{OP} containing the value of an efficient operation for every triplet (v, w, S) can be computed as follows:

$$D_{\text{OP}}(S,v,w) = \begin{cases} D_{\text{T}}(S,v,w) & \text{if } S = \{v,w\} \\ \infty & \text{if } v \notin S \vee w \notin S \\ \min\limits_{d \in S \setminus \{v,w\}} \max\left\{c^d(v,d) + c^d(d,w), D_{\text{T}}(S \setminus \{d\}, v, w)\right\} & \text{otherwise} \end{cases} \tag{3}$$

When we assume that all truck tours D_{T} have been computed prior to the construction of D_{OP}, we can compute the D_{OP} table in $O(2^n \cdot n^3)$ time, as there are $2^n \cdot n^2$ sub-problems, that all can be solved in $O(n)$ time.

In the third and final pass, we compute the optimal TSP-D paths in a table D using the same sub-problems as those of the regular TSP in Equation 1.In a sub-problem $D(S,v)$ we look for a minimum cost sequence of efficient operations that starts at v_0, ends at v and visits all locations in S. Let us now consider the final operation of such a minimum cost sequence, which begins at some location $u \in S$, ends at v and visits locations $T \subseteq S$. Clearly, the sequence of operations prior to this operation must be of minimum cost, start at v_0, end at u and visit locations $S \setminus (T \setminus \{u\})$. Thus, the sub-problem $D(S,v)$ can be solved by combining operations that end at v and cover a subset of S with the solution to a smaller sub-problem. The relation between these sub-problems can be expressed as follows:

$$D(S,v) = \begin{cases} 0 & \text{if } S = \{v_0\}, v = v_0 \\ \infty & \text{if } v \notin S \\ \min\limits_{w \in S} \min\limits_{T \subset S} D(S \setminus T, w) + D_{\text{OP}}(T,w,v) & \text{otherwise} \end{cases} \tag{4}$$

A straightforward analysis of the runtime of this algorithm tells us that there are $2^n \cdot n$ sub-problems and that we have to do at most $2^n n$ work to solve a sub-problem (assuming the smaller sub-problems have been solved), resulting in a runtime of $O(4^n \cdot n^2)$. However, with a more careful analysis we can see the algorithm actually performs better. To prove this, we apply the binomial theorem, which reads:

Theorem 3.1 (Binomial Theorem, folklore)

$$(x+y)^r = \sum_{k=0}^{r} \binom{r}{k} x^k y^{r-k} \tag{5}$$

Theorem 3.2 *The TSP-D with n locations can be solved in $O(3^n n^2)$ time using dynamic programming.*

Proof In Equation 4 we can see that there are $\binom{n}{1}$) sub-problems where $|S| = 1$, $2\binom{n}{2}$ sub-problems where $|S| = 2$ and in general at most $n\binom{n}{i}$ sub-problems where $|S| = i$. If

we know that a certain sub-problem has an S with $|S| = i$, we can see that the amount of work we need to do to solve it is $n \cdot 2^i$, as we are only considering subsets of S. Putting these two facts together we can apply the binomial theorem to obtain a tighter analysis of the runtime:

$$\sum_{i=0}^{n} \binom{n}{i} 2^i n^2 = n^2 \sum_{i=0}^{n} \binom{n}{i} 2^i \cdot 1^{n-i} \tag{6}$$

$$= 3^n \cdot n^2 \tag{7}$$

Here we first include a factor 1^{n-i} in the analysis, which allows us to apply the binomial theorem to the result. We have three passes needed to compute the final TSP-D tour, where the first pass and second pass both take $O(2^n \cdot n^3)$ time. However, we can rewrite $O(3^n)$ as $O(2^n \cdot \left(\frac{3}{2}\right)^n)$ and argue that a factor n grows asymptotically slower than $\left(\frac{3}{2}\right)^n$. As a result, the $O(3^n n^2)$ running time is the dominant one among the three pass. Pseudo code that implements this approach is presented in Algorithm 2

Algorithm 2: Algorithm to compute an optimal solution to the TSP-D

Data: A set of locations V, a depot v_0 and precomputed table D_{OP}

Result: A table $D(S, v)$ with optimal TSP-D paths that starts at the depot v_0, end at location v and covers all locations in S.

```
1 D({v_0}, v_0) ← 0 ;
2 for i = 1, ..., |V| do
3     for S ⊆ V where |S| = i do
4         for T ⊆ V \ S do
5             for u, w ∈ V^2 do
6                 z ← D(S, u) + D_OP(T ∪ {u, w}, u, w) ;
7                 if z < D(S ∪ T, w) then
8                     D(S ∪ T, w) ← z ;
```

One advantage of the dynamic programming approach is that practical restrictions on the feasibility of different operations can easily be incorporated by forbidding infeasible operations in the first and second pass by setting their costs to infinity. This means we can ignore these operations when computing a minimum cost sequence of operations.

A disadvantage of the approach is that practical running times and storage requirements prevent solving larger problems. In the next subsections, we discuss two ways to speed up the running times of the algorithm. In section 3.2, we discuss heuristically restricting the number of operations and in section 3.3 guiding the search for the optimal path in the final pass of the algorithm.

3.2 Restricting the number of operations

As the operations can involve an arbitrary number of nodes this results in an exponential number of sets. Instead of creating all possible drone operations, we may limit the set of operations to reduce the number of sub-problems we need to consider in the first and second passes. This directly influences memory requirements and the running times of the algorithm. In particular, we consider restricting the number of truck nodes per operation to $k < |V|$. This helps reduce the amount of required work in all passes of the algorithm.

It seems reasonable to assume that the number of truck nodes per operation in most good solutions for most instances will be relatively small. This is especially the case if the drone travels faster than the truck. That is, fewer truck nodes per operation implies that more work is preformed by the drone and thus the advantage of parallelization is greater.

Unfortunately, restrictions on the number of truck nodes may prevent finding an optimal solution. That is, it is not difficult to construct a problem instance in which the optimal solution consists of just one operation with many truck nodes. This is for example the case if all but one node (the designated drone node) are close to the depot, while the designated drone node is located sufficiently far from all other nodes.

Theorem 3.3 *For any instance of the TSP-D with symmetric drone costs, i.e.* $c^d(v, w) = c^d(w, v)$*, there is a TSP-D tour without truck nodes (that is, every operation consists of a start node, an end node, and possibly a drone node), whose time duration is at most twice the time duration of the optimal TSP-D tour for that instance.*

Proof Let $\hat{O} = (\hat{o}_1, \hat{o}_2, \ldots, \hat{o}_l)$ be an optimal TSP-D tour for the considered instance. We construct a TSP-D tour without truck nodes from $\hat{O}$ by replacing each operation $o = \hat{o}_i$ for $i = 1, \ldots, l$ by a sequence of operations without truck nodes with at most double completion time.

Let o be an operation with k truck nodes $v^1, \ldots, v^k$, start node v^0, end node v^{k+1}, and drone node v^d. If $c^d(v^0, v^d) \leq c^d(v^d, v^{k+1})$, we replace o by the sequence $(o^0, o^1, o^2, \ldots, o^{k+1})$ where in operation o^0 the drone flies from v_0 to v^d while the truck stays at v^0 and in o^j for $j = 1, \ldots, k+1$ the truck drives from v_{j-1} to v_j (without the drone leaving the truck). Otherwise, we replace o by the sequence $(o^1, o^2, \ldots, o^{k+1}, o^0)$ where in operation o^0 the drone flies from v^{k+1} to v^d while the truck stays at v^{k+1} and o^j for $j = 1, \ldots, k+1$ as above. In both cases we obtain

$$\sum_{j=0}^{k+1} t(o^j) = \underbrace{2c^d(v_0, v_d)}_{\leq c^d(o)} + \underbrace{\sum_{j=0}^{k+1} c(v_{j-1}, v_j)}_{=c(o)} \leq 2 \cdot \max\{c(o), c^d(o)\} = 2t(o).$$

We can see that this bound is tight from the example in Figure 2, where we have a depot and two customers v_1 and v_2. In this example, we assume that the speed of the

drone is equal to the speed of the truck and both customers have the same travel time from the depot. In the optimal solution, one node is served by the drone and one by the truck as both vehicles start and end at the depot. As the drone and the truck can serve the nodes simultaneously, this gives a solution value of 2. In case truck nodes are not allowed in an operation, it is not possible to parallelize the deliveries. This means that in the optimal restricted solution, we either serve the nodes sequentially with one of the vehicles or the truck has to wait for the drone at one of the locations. Both cases result in a solution value of 4. In the numerical experiments in Section 4, we evaluate the impact of these restrictions in more practical instances than the presented example.

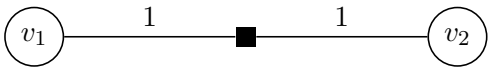

Figure 2: Two customers (circle) that need to be served from the depot (rectangle). Assuming that the truck and drone are equally fast, the optimal solution in case no truck-only nodes are allowed has twice the costs of a solution without restrictions.

It is relatively straightforward to adopt our DP-approach to take into account these restrictions, as we need to consider only truck paths of at most $2 + k$, as the start and end nodes of an operation do not count as truck nodes. Thus, we need to only compute the solution to the sub-problems $D_{\text{T}}(S, v, w)$ where $|S| \leq 2 + k$. When we compute the table $D_{\text{OP}}(S, v, w)$, we must take care to consider only the sub-problems where $|S| \leq k + 2$ if $v \neq w$ and $|S| \leq k + 1$ otherwise.

This implies that for the first two passes we get $O(\sum_{i=1}^{k+2} i^2 \binom{n}{k})$ sub-problems rather than $O(n^2 \cdot 2^n)$ for the unrestricted case. For the third pass, we unfortunately still need to consider all $O(2^n \cdot n)$ sub-problems, but the amount of work required to solve each sub-problem can be reduced: instead of considering all subsets of S we only need to consider subsets T of size $k+2$, rather than all subsets of S. If for example $k = 0$, this implies that every sub-problem can be solved in $O(n^{k+2})$ time and thus all solutions can be computed in $O(2^n \cdot n^{k+3})$ time, rather than in $O(3^n n^2)$ time.

3.3 A* approach

One possible disadvantage of dynamic programming approaches, is that their run time if often very consistent: even if an instance clearly has many sub-problems that are not relevant to the *optimal* solution, they are still solved. For this reason we also consider an approach that attempts to ignore irrelevant sub-problems, based on ideas from *informed search*. In informed search algorithms, one typically searches through a graph of (partial) solutions to sub-problems, often referred to as states, exploiting additional information about the state space to direct the search. The states in such a search problem are connected to each other by the same recurrence relationship that connect the sub-problems,

in such a way that an arc which goes from state a to b has costs equal to the difference of the sub-problems corresponding to states b and a. If we take the relationship of Equation 4 as an example, and arc from (S, v) to (T, w) would have costs $D_{\text{OP}}(T \setminus S, v, w)$.

By only generating states if they seem relevant to finding a good solution, it may not be necessary to solve all $2^n \cdot n$ sub-problems, but significantly less. The idea is to heuristically estimate how good a partial solution is, and guide the search toward the path to the optimal solution. This way we potentially limit the number of states that needs to be considered in the third pass of the algorithm.

One well known algorithm within the scope of *informed search* is the A^* algorithm, first introduced by Hart et al. (1968, 1972). Let us consider a graph based on Equation 4. A^* can be regarded as an algorithm looking for the shortest path in this graph from the starting state $(\{v_0\}, v_0)$ to the goal state (V, v_0). The key idea is that A^* introduces an *admissible* heuristic function l that provides a lower bound on the path from a given state to the goal state.

By only expanding states for which the sum of the path to that state and the value of the heuristic function is minimal, the algorithm does not generate states that are far from the optimal path to the goal state. It was proven by Dechter and Pearl (1985) that if the heuristic function l never over-estimates the distance to the goal state, this approach finds the optimal solution.

In Agatz et al. (2017), we show that the minimum spanning tree provides a lower bound on the costs of the TSP-D, for instances where the triangle inequality holds.

Theorem 3.4 *(Agatz et al., 2017) A solution $(\mathcal{R}, \mathcal{R})$, consisting of a TSP tour $\mathcal{R}$ constructed with the minimum spanning tree heuristic is a $(2 + \alpha)$-approximation for the TSP-D.*

Based on Theorem 3.4, we can compute a lower bound using a minimum spanning tree which spans the last location visited in our current state, the locations that are not covered yet, and the depot. This lower bound function can be provided for the pseudo-code of an A^* algorithm for the TSP-D as presented in Algorithm 3.

4 Numerical experiments

Using the uniform instances presented in Agatz et al. (2017) and several new ones generated using the same approach, we conduct various experiments. The instances consist of points uniformly distributed in a 100×100 two dimensional square with the depot at one of the corners, and we take the Euclidian distance between pairs of locations. As a consequence the triangle inequality holds, so we can use the minimum spanning tree lower bounds for the A* algorithm. With these instances, we assess our exact dynamic programming approaches: the basic approach (DP) and the A*. Moreover, we also evaluate the

Algorithm 3: Outline of the A* algorithm for the TSP-D

Data: A set of locations V, a depot v_0, a precomputed table D^k_{OP}, a restriction on truck-only nodes k, a lower bound function l

Result: The optimal TSP-D tour that can be constructed using only the operation in D^k_{OP}

```
1  Q ← new Priority Queue ;
2  P ← ∅ ;
3  Add state ({v_0}, v_0) with costs 0 and key l(V) to Q ;
4  while Q not empty do
5      Remove state (S, u) with costs c and minimum key from Q ;
6      Add (S, u) to P ;
7      if S = V ∧ u = v_0 then
8          return Backtrack path of operations to state (S, u)
9      foreach w ∈ V do
10         foreach operation o ∈ D^k_OP starting in w ending in u with no truck-only or
             drone-only nodes in S, costs c_o and covering nodes V_o do
11             s' ← (S ∪ V_o ∪ {w}, w) ;
12             if s' not in P then
13                 Add s' with costs c + c_o and key c + c_o + l(s') to Q ;
```

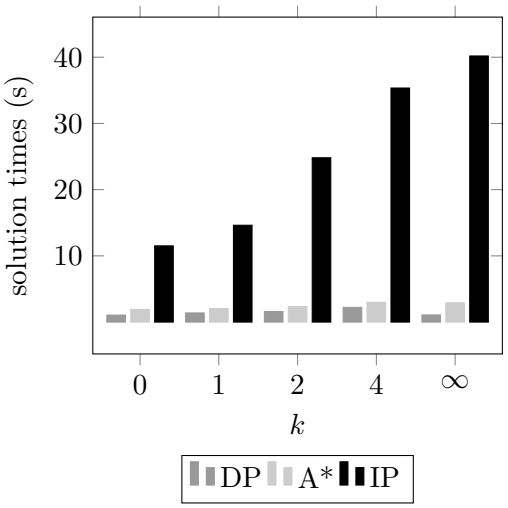

Figure 3: Solution times for different approaches and different number of allowed truck nodes per operation k. Averages over 10 instances with 10 nodes.

impact of restricting the number of truck nodes per operation k, i.e. 0, 1, 2, 4. We will refer to the unrestricted case as $k = \infty$. All experiments assume that the drone is twice as fast as the truck ($\alpha = 2$).

Experiments were executed on the Lisa computer cluster of SURFsara. During each run, 10 instances were solved in parallel by a cluster node equipped with an Intel® Xeon® E5-2650 v2 CPU. Depending on the size of the instances solved, nodes with either 32GB or 64GB of RAM were used, with either 2GB or 6GB of RAM assigned per instance. The computer cluster runs on Debian Linux. The algorithms were implemented in Java, and executed using the Oracle JDK version 1.8.0_40 on the cluster. The IP was solved using CPLEX 12.6.3. Minimum spanning trees were computed using Kruskal's algorithm on the Delaunay triangulation of the geometric instances. The Delaunay triangulations were computed using the Java Topology Suite version 1.13.

In the first set of experiments, we compare the running times of the DP and the A* for different restrictions. As an additional benchmark, we also compare the results to the IP as presented in Agatz et al. (2017) that uses binary decisions variables for each feasible drone operation and constraints that ensure that the set of operations in the solution represents can be converted to a TSP-D tour that covers all nodes. These constraints include sub-tour elimination constraints for every subset of locations. In the unrestricted case $k = \infty$, a variable for all efficient operations must be added, all of which can be generated by the first two passes of our dynamic programming case. For restricted cases, very operations are considered and thus fewer variables need to be added to the model. Unfortunately, the number of sub-tour elimination constraints does not increase when the number of truck nodes is restricted.

Figure 3 provides the solution times of the different approaches for the uniform instances with 10 nodes from Agatz et al. (2017). We see that the dynamic programming approaches consistently outperforms the IP. The main reason for this bad performance is that the IP contains not only an exponential number of variables, obtained by passes one and two of the DP, but also an exponential number of constraints ($n \cdot 2^n$ to be precise). This results in a large IP, even if the number of operations is reduced.

Comparing the running types of the dynamic programming approaches, we see that the DP is faster than the A*. This suggest that savings of potentially finding the optimal solution faster do not offset the costs for the numerous additional lower bound calculations in the A* for these instances.

Table 2 presents the running times for larger instances of up to 20 nodes. As expected, the results show that restricting the number of allowed truck nodes per operation k significantly reduces the running times. In all but one set of instances, we see that the running times strictly increase with k. As an example, we see that it takes less than 30 minutes to solve the $n = 20$ instances with $k = 0$ while it takes more than 10 hours to solve the same instances with $k = 2$.

For the instance with ten nodes, however, the unrestricted cases ($k = \infty$) have slightly smaller running times, on average, than the approaches with $k = 4$. The reason for this is that our implementation uses different data structures to build the table of operations created in passes one and two depending on whether or not restrictions are in place. For the unrestricted case it is relatively straightforward to work with an arrays indexed by bitwise representations of the sets. In case of restrictions this approach breaks down and hash-based sparse data structures were used. These sparse data structures have more overhead and as the restrictions do not eliminate many operations in the smaller instances, the savings obtained from building fewer operations are less than the additional costs due to the overhead of the hash-based data structure.

Comparing between the different exact approaches, we see that the DP is faster for smaller instances and A* is faster for larger instances in all cases with restrictions. However, in the unrestricted $k = \infty$ cases the DP always outperforms the A*. A possible explanation for this, is that without restrictions there is already an exponential number of operations ($O(2^n n)$) to consider in the first step. As a consequence, all relevant subproblems are immediately generated. Furthermore, the lower bound for each subproblem is immediately computed, resulting in a lot of overhead compared to the DP. For cases with restrictions, fewer search states are reachable from the initial search state and as a result, we observe that A* performs better than DP starting at instances of size 14 with $k = 0$, while it is already faster for instances of size 11 with $k = 4$.

Table 3 shows the impact of the restrictions on the solution quality. Therefore, we compare the optimal solutions of the approaches that allow at most k truck nodes per

k	algo	10	11	12	13	14	15	16	17	18	19	20
0	DP	1	2	4	7	17	44	1:55	4:60	12:60	33:10	1:24:45
0	A*	3	3	6	7	11	29	1:15	1:35.3	4:59	10:39	27:29
1	DP	2	3	8	20	59	2:55	8:37	24:36	1:08:52	3:18:19	9:09:31
1	A*	2	5	8	16	26.9	1:32	4:21	6:28	21:28	55:34	2:08:04
2	DP	2	5	16	52	2:50	9:32	31:06	1:42:58	5:25:04	*	*
2	A*	3	5	10	27	1:00	4:21	13:47	24:40	1:24:06	4:14:57	10:35:26
4	DP	3	9	38	3	13:07	1:00:28	4:29:48	*	*	*	*
4	A*	4	8	19	1:14	3:47	22:05	1:33:51	4:10:13	10:13:05	*	*
∞	DP	2	5	16	1:24	8:00	50:43	5:29:28	*	*	*	*
∞	A*	4	8	25	1:59	8:43	1:13:39	8:00:01	*	*	*	*

* the algorithm did not find a solution within 12 hours

The grey cells indicate the smallest average running time for a specific set of instances

Table 2: Solution times (hours:minutes:seconds) for different solution approaches for different numbers of allowed truck nodes per operation k, uniformly distributed, $\alpha = 2$, averages over ten instances

operation Z^k with the optimal solution for the problem without restrictions Z^∞, i.e., $\frac{Z^k - Z^\infty}{Z^\infty} \times 100$. We provide the following three measures for the solution quality:

- Δ %: the average relative deviation from the optimal solution Z^∞
- max %: the max relative deviation from the optimal solution Z^∞
- # opt: the number of times that the algorithm with restrictions finds the optimal solution Z^∞

As expected, we see that the solution quality improves with k. The worse performance is associated with the approach that does not allow any truck nodes ($k = 0$). Here, we see an average optimality gap ranging from 2.4 percent to 5.9 percent with a maximum gap of 11 percent. While this is a substantial gap, it much better than the theoretical worse-case gap of 50 percent from Theorem 3.2. We see that the maximum optimality gap quickly goes down to 5.6 percent for $k = 1$ and only 2.8 percent for $k = 2$. For $k = 4$, we find the optimal solution in 69 of the 70 instances and have a maximum gap of less than 1 percent.

Moreover, We see that the performance is not that sensitive to the size of the instance n. This suggests that the costs of not allowing large operations with many truck nodes is relatively small, even for larger instances. One potential explanation for this is that while the larger instances may potentially involve larger operations, there are also many more good solutions possible that involve less truck nodes.

	$k=0$			$k=1$			$k=2$			$k=4$		
n	Δ %	max %	# opt	Δ %	max %	# opt	Δ %	max %	# opt	Δ %	max %	# opt
10	2.4	6.4	2/10	0.4	2.7	6/10	0.0	0.2	9/10	0.0	0.0	10/10
11	3.1	10.1	3/10	1.1	5.6	7/10	0.0	0.0	10/10	0.0	0.0	10/10
12	4.8	10.3	1/10	7.0	3.8	2/10	0.4	2.2	6/10	0.0	0.0	10/10
13	3.3	6.0	0/10	1.3	1.1	7/10	0.0	0.0	10/10	0.0	0.0	10/10
14	3.6	7.7	1/10	0.2	3.5	3/10	0.1	0.6	9/10	0.0	0.0	10/10
15	4.7	10.3	0/10	1.0	3.6	5/10	0.2	1.7	8/10	0.1	0.6	9/10
16	5.9	11.0	0/10	1.0	2.8	3/10	0.4	2.8	7/10	0.0	0.0	10/10

Table 3: Solution quality for different numbers of allowed truck nodes per operation k, uniformly distributed, $\alpha = 2$, averages over ten instances

To provide more insight into the characteristics of the solutions for different truck node restrictions, we compare the total number of drone and truck nodes in the solutions as a percentage of the total number of nodes n in the instance in Figure 4. Overall, we see that the number of drone nodes is much higher than the number of truck nodes. The reason for this is that our drone is twice as fast as the truck, which allows the drone to meet up with the truck at his next stop without leaving time for the truck to serve more nodes.

Looking at the impact of k, we see that the number of drone nodes increases when allowing less truck nodes. This is intuitive as larger operations with one drone and several truck nodes are replaced by several smaller operations.

Figure 5 provides a different way of looking at this behavior. Here we see the total waiting times (summed over all operations) of the truck and the drone relative to the total time required to serve all nodes. The results show that the drone generally incurs more waiting time than the truck. This again is related to the fact that our drone is faster than the truck. As expected, we see that the waiting time of the truck goes up if we restrict the maximum number of truck nodes.

5 Conclusions and future research

We show that by using dynamic programming, we can solve larger problems than with the mathematical programming approaches that have been presented in the literature so far. Moreover, we show that restrictions on the number of operations can help significantly reduce the solution times while having relatively little impact on the overall solution quality.

While we have considered restricting the number of truck nodes per operation to heuristically reduce the running times of our algorithm, future research could investigate

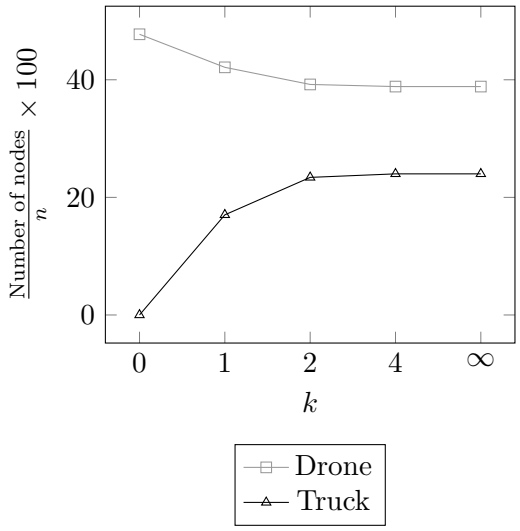

Figure 4: Number of truck and drone nodes in the optimal solutions for different maximum numbers of truck nodes per operation k, averages over 60 instances of size (n) 10 to 15

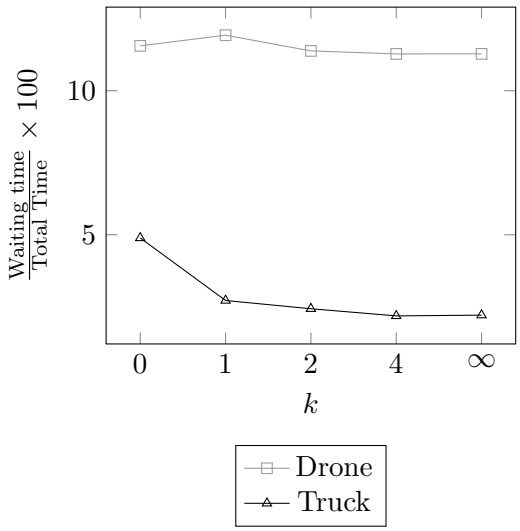

Figure 5: Waiting time of truck and drone in the optimal solutions for different maximum numbers of truck nodes per operation k, averages over 60 instances of size (n) 10 to 15

other dynamic programming based heuristics. One promising direction for future research is to study different ways to identify 'bad' operations upfront, e.g. in which either the drone or the truck incurs a lot of waiting time.

Also, we have considered simple operations with at most one drone. If we assume that all drones in an operation must start and end at the same node, it is straightforward to include multi-drone operations in the second pass of our dynamic programming approach. However, it is not that clear how to incorporate multiple drones if each drone can start and end at different nodes. This is an interesting area for future research.

References

Niels Agatz, Paul Bouman, and Marie Schmidt. Optimization approaches for the traveling salesman problem with drone. *Transportation Science, Forthcoming*, 2017.

David L. Applegate, Robert E. Bixby, Vašek Chvátal, and William J. Cook. *The traveling salesman problem: a computational study*. Princeton University Press, 2011.

Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM (JACM)*, 9(1):61–63, 1962.

J.G. Carlsson and S. Song. Coordinated logistics with a truck and a drone. *Management Science, Forthcoming*, 2017.

Jamie Condliffe. Drones get set to piggyback on delivery vans. *Technologyreview*, 2016. URL `https://www.technologyreview.com/s/602316/drones-get-set-to-piggyback-on-delivery-vans/`.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009. ISBN 0262033844, 9780262033848.

Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of a*. *Journal of the ACM*, 32(3):505–536, 1985.

Sergio Mourelo Ferrandez, Timothy Harbison, Troy Weber, Robert Sturges, and Robert Rich. Optimization of a truck-drone in tandem delivery network using k-means and genetic algorithm. *Journal of Industrial Engineering and Management*, 9(2):374–388, 2016.

Quang Minh Ha, Yves Deville, Quang Dung Pham, and Minh Hoàng Hà. Heuristic methods for the traveling salesman problem with drone. *arXiv preprint arXiv:1509.08764*, 2015.

P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107, July 1968. ISSN 0536-1567. doi: 10.1109/TSSC.1968.300136.

Peter E Hart, Nils J Nilsson, and Bertram Raphael. Correction to a formal basis for the heuristic determination of minimum cost paths. *ACM SIGART Bulletin*, (37):28–29, 1972.

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

A Leendert Kok, C Manuel Meyer, Herbert Kopfer, and J Marco J Schutten. A dynamic programming heuristic for the vehicle routing problem with time windows and european community social legislation. *Transportation Science*, 44(4):442–454, 2010.

Chryssi Malandraki and Mark S Daskin. Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms. *Transportation science*, 26(3):185–200, 1992.

Chryssi Malandraki and Robert B Dial. A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem. *European Journal of Operational Research*, 90(1):45–55, 1996.

Aristide Mingozzi, Lucio Bianco, and Salvatore Ricciardelli. Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints. *Operations Research*, 45(3):365–377, 1997.

Chase C. Murray and Amanda G. Chu. The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. *Transportation Research Part C: Emerging Technologies*, 54:86–109, 2015.

Stefan Poikonen, Xingyin Wang, and Bruce Golden. The vehicle routing problem with drones: Extended models and connections. *Networks*, 70(1):34–43, 2017. ISSN 1097-0037. doi: 10.1002/net.21746. URL `http://dx.doi.org/10.1002/net.21746`.

Andrea Ponza. Optimization of drone-assisted parcel delivery. 2016.

Harilaos N Psaraftis. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2):130–154, 1980.

Jack Steward. A drone-slinging ups van delivers the future. *Wired*, 2017. URL `https://www.wired.com/2017/02/drone-slinging-ups-van-delivers-future/`.

Xingyin Wang, Stefan Poikonen, and Bruce Golden. The vehicle routing problem with drones: several worst-case results. *Optimization Letters*, pages 1–19, 2016.