Dynamic Programming Approaches for the Traveling Salesman Problem with Drone

Article *in* SSRN Electronic Journal · January 2017

DOI: 10.2139/ssrn.3035323

CITATIONS
53

READS
3,964

3 authors, including:

Paul Bouman
Erasmus University Rotterdam
12 PUBLICATIONS **876** CITATIONS
SEE PROFILE

Marie Schmidt
University of Wuerzburg
67 PUBLICATIONS **1,827** CITATIONS
SEE PROFILE

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 truck and a drone gives rise to a new planning problem that is known as the Traveling Salesman Problem with Drone (TSP-D). This paper presents exact solution approaches for the TSP-D based on dynamic programming and provides an experimental comparison of these approach. 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 locations the truck can visit while the drone is away 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).

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 needs to be synchronized with that of the truck. Figure 1 shows an example solution of the TSP-D.

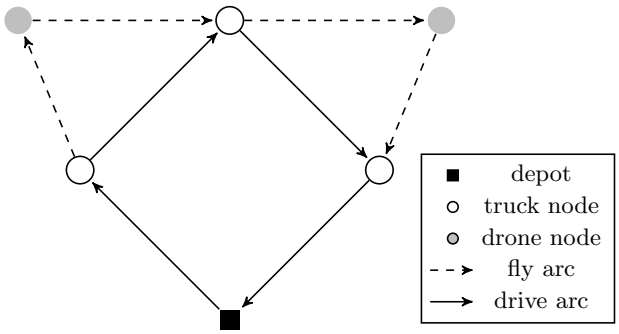


Figure 1: An example TSP-D solution (Agatz et al., 2017)

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. Agatz et al. (2017) experimentally analyze the time savings of using truck and drone instead of using truck only. 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 up to now, most research has primarily 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), dynamic programming approaches were first proposed in Held and Karp (1962); Bellman (1962). 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

Type of approach	Solution procedure
Exact methods	Integer linear programming (Agatz et al., 2017)
	Dynamic programming (this paper)
Approximation algorithms	Improvements starting from TSP-tour (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

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 3-pass dynamic programming approach to solve the TSP-D problem and extend the last pass of this approach to an A* algorithm. Furthermore, we apply these exact algorithms to the problem where we restrict the number of locations the truck may visit while separated from the drone. These restrictions result in shorter computation times, but come at the cost of potentially removing the optimal solution from the solution space. We evaluate the different approaches in an extensive computational study comparing run times and solution quality.

The remainder of this paper is organized as follows. In Section 2, we formally define the TSP-D. In Section 3, we develop 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. This assumption is common in the literature, and can be justified by the fact that a) launching and receiving

drones on a moving truck is technologically still very challenging (Orphanides, 2016) and b) parking at intermediate points on the route may not always be possible.

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 $v, w \in V$. The special case where $c^d(v, w) = \alpha c(v, w)$ 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. If the operation contains a drone node, 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. The truck can travel directly from the start node to the end node or can visit any number of truck nodes in between. When the start node is identical to the end node and the operation does not contain any truck nodes, the truck waits at the start/end node for the drone to deliver the package and return. If the operation does not contain a drone node, the drone travels on the truck directly from start node to end node.

Figure 2 provides an example of an operation (Agatz et al., 2017). 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 drone 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.

3 Solution approaches

In this section we present dynamic programming approaches for the TSP-D based on the well-known Bellman-Held-Karp dynamic programming algorithm for the TSP (Held and Karp, 1962; Bellman, 1962). Dynamic Programming can be used to find optimal solutions to problems, if the optimal solutions have an *optimal substructure*, i.e., if they can be

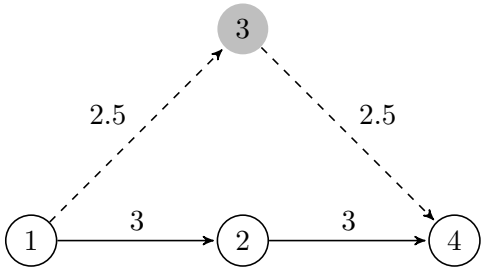


Figure 2: An operation with a combined node serving as start node (1), a combined node serving as end node (4), a drone node (3) and a truck node (2)

split up in an optimal solution to a smaller sub-problem and some contribution that is independent from the solution to the smaller subproblem.

Our approaches consist 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 that represent the least costly way to cover a set of nodes with an operation.

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

We now first summarize the the well-known Bellman-Held-Karp dynamic programming algorithm for the TSP (Held and Karp, 1962; Bellman, 1962) in Section 3.1.

Afterwards, we describe how we modify this procedure to execute the three passes. The first and second pass are described in Sections 3.2 and 3.3. For the third pass, we describe the general idea in Section 3.4 and two different variants of how to implement it in Sections 3.4.1 and 3.4.2. Furthermore, we discuss to restrict the number of nodes in an operation in Section 3.5.

3.1 Dynamic programming approach for the standard TSP

The optimal solution for the regular TSP can be considered as a path that starts at an arbitrary origin point v, visits all locations in V and ends at v. The costs of the final solution, which can be distance, time, emissions, or anything else that can be modelled as the sum of individual steps, is denoted as $D_{\text{TSP}}(V, v)$

For a set $S \subset V$ that contains v and a node $w \in S \setminus \{v\}$ let $D_{\text{TSP}}(S, w)$ be the costs of the shortest path that starts at the origin point, visits all locations in S and ends at location w. The shortest path of this sub-problem consists of two parts: the last arc on the path that goes from some location $u \in S$ to w, and a shortest path that starts at the

origin point v, visits all locations in $S \setminus \{w\}$, and ends in u. If the path for the full set S does not contain the shortest sub-path for $S \setminus \{w\}$, 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 $D_{\text{TSP}}(S, w)$ can be solved by considering all arcs $(u, w) : u \in S \setminus \{w\}$, and adding each of those arcs to the solution for the smaller sub-problem $D_{\text{TSP}}(S \setminus \{w\}, u)$.

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:

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

As there 2^n possible subsets of V and n locations in v, we can see that there are at most $n \cdot 2^n$ sub-problems to solve here. 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.

3.2 First pass

As the first pass of our three-pass approach, we adapt the dynamic programming approach for the regular TSP to find the shortest path for the truck for every start node v, end node w, and set of truck nodes S covered by the path. Each of these sub-problems can be solved based on smaller sub-problems using the same idea as for the regular TSP:

$$D_{\text{T}}(S, v, w) = \begin{cases} \infty & \text{if } w \notin S \\ c(v, w) & \text{if } S = \{w\} \\ \min\limits_{u \in S} \{D_{\text{T}}(S \setminus \{w\}, v, u) + c(u, w)\} & \text{otherwise} \end{cases} \tag{2}$$

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.

3.3 Second pass

In the second pass, we combine the truck paths with drone nodes to obtain efficient operations, that is operations that represent the least costly way to cover a set of nodes S with an operation for given start node v and end node w. An optimal TSP-D tour can be

Algorithm 1: Dynamic Programming algorithm for the original TSP

Data: A set of locations V, an arbitrary location $v \in V$ 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 locations ;
3  Initialize v as an arbitrary location in V ;
4  foreach w ∈ V do
5      D_TSP({w}, w) ← c(v, w) ;
6      P_(({w}, w) ← v ;
7  for i = 2, ..., |V| do
8      for S ⊆ V where |S| = i do
9          foreach w ∈ S do
10             foreach u ∈ S do
11                 z ← D_TSP(S \ {w}, u) + c(u, w) ;
12                 if z < D_TSP(S, w) then
13                     D_TSP(S, w) ← z ;
14                     P(S, w) ← u ;
15 return path obtained by backtracking over locations in P starting at P(V, v) ;
```

constructed using *efficient* operations only as a TSP-D tour will not become longer when an inefficient operation is replaced by an efficient operation.

To find efficient operations associated with every triplet (S, v, w), where the operation starts at location v, ends at location w and visits all locations in S, we expand the table of truck paths to create a table of operations, by adding the drone movement on top of the truck paths. 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 node d 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 (S, v, w) can be computed as follows:

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

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 at most $2^n \cdot n^2$ sub-problems, that all can be solved in $O(n)$ time.

3.4 Third pass

In the third pass, we compute the optimal sequence of efficient operations that covers all locations and starts and ends at the depot. To do this, we iteratively solve the subproblem of finding the best sequence of operations starting at the depot, covering set of nodes S, and ending in node w. That is: w is the end node of the last operation in the sequence, we call it *sequence end node* in the following.

In Sections 3.4.1 and 3.4.2 we present two variants of this approach which we compare later in our numerical experiments. These variants can be considered as different methods to find a shortest path from source to sink in a so-called *state-graph*. In a *state-graph*, vertices represent subproblems (called *states*) and the arcs correspond to dependencies between states in the computation.

Thus, in the third pass of our algorithm, each vertex corresponds to a pair (S, w) of locations already covered in our sequence of operations and sequence end location w. We aim to find a shortest path from the source state $(\{v_0\}, v_0)$ where we have not covered any locations yet and start from the depot v_0 to the sink state (V, v_0) which covers all locations and ends in the depot v_0.

The arcs of the graph represent going from one state to another by executing one more operation. That is, state (S_1, w_1) is connected to state (S_2, w_2) by an arc if $S_1 \subsetneq S_2$. The cost of the arc is the cost of an efficient operation with start node w_1, end node w_2 covering nodes $S_2 \setminus S_1$. This cost is given by $D_{OP}(S_2 \setminus S_1, w_1, w_2)$ which we compute in the second pass. The structure of this state-graph is visualized in greater detail in Section 3.5, where we consider how restrictions of the number of truck-only nodes in an operation affect the structure of the state-graph.

Finding an optimal TSP-D tour now corresponds to finding a shortest path from source $(\{v_0\}, v_0)$ to sink (V, v_0) in this state-graph with respect to the described arc costs. We present two different approaches to find such a shortest path. The first implementation corresponds to a 'standard' dynamic programming approach, while the second one corresponds to a A^* algorithm Hart et al. (1968, 1972). We describe these two approaches in more detail below.

3.4.1 Standard dynamic programming implementation of third pass

We compute the costs of all states using a table D. To compute the costs of a state (subproblem) $D(S, w)$ we look for a minimum cost sequence of efficient operations that starts at v_0, ends at w and visits all locations in S. This is specified in (4), where we compute the cost of a state as the minimum over the cost of possible preceding states plus the arc costs of the connecting arc connecting the two states (which is the cost of the corresponding efficient operation).

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

A straightforward analysis of the runtime of this algorithm tells us that there are at most $2^n \cdot n$ sub-problems and that we have to do at most $2^n \cdot n$ 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.*

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

Data: A set of locations V and a depot v_0 **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.

```
D ← ∞ ;
D(∅, v_0) ← 0 ;
Precompute D_OP
;
for i = 1, ..., |V| do
    for U ⊆ V where |U| = i do
        for T ⊆ V \ U do
            for (u, w) ∈ U × V do
                z ← D(U, u) + D_OP(T ∪ {w}, u, w) ;
                if z < D(U ∪ {u, w} ∪ T, w) then
                    D(U ∪ {u, w} ∪ T, w) ← z ;
return D ;
```

Proof In Equation 4 we can see that there are $O(\binom{n}{1} \cdot n)$ sub-problems where $|S| = 1$, $O(\binom{n}{2} \cdot n)$ sub-problems where $|S| = 2$ and in general at most $O(\binom{n}{i} \cdot n)$ 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 $O(n \cdot 2^i)$ as there are 2^i possible subsets T of S and $O(n)$ possible choices for $u \in S$. As there are $O(\binom{n}{i} \cdot n)$ sub-problems for a particular i, and each of those requires $O(2^i \cdot n)$ work to solve, we can solve *all* sub-problems for a particular i in $O(\binom{n}{i} 2^i n^2)$ time. In order to solve all sub-problems, we need to solve the sub-problems for every i from 1 to n. Since we over-estimate the number of sub-problems when we let i range from 0 to n rather than 1 to n, we can safely apply the binonial theorem:

$$O\left(\sum_{i=0}^{n} \binom{n}{i} 2^i n^2\right) = O\left(n^2 \sum_{i=0}^{n} \binom{n}{i} 2^i \cdot 1^{n-i}\right) \tag{6}$$

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

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 $O(2^n \cdot \left(\frac{3}{2}\right)^n)$. As a factor n grows asymptotically slower than $\left(\frac{3}{2}\right)^n$, the $O(3^n n^2)$ running time is the dominant one among the three passes. Pseudo code that implements this approach in a bottom-up fashion is presented in Algorithm 2

3.4.2 A^* implementation of third pass

One possible disadvantage of the approach described in Section 3.4.1 is that even if an instance has many sub-problems that are clearly not relevant to the *optimal* solution, they are still solved. For this reason, we also consider an approach that attempts to skip irrelevant sub-problems, based on ideas from *informed search*. By only generating states that 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 estimate how good a partial solution is, and guide the search towards the optimal path. This way, we potentially limit the number of sub-problems that needs to be considered in the third pass of the algorithm.

We do this by executing an A^* algorithm Hart et al. (1968, 1972) to find a shortest path from source $(\{v_0\}, v_0)$ to sink (V, v_0) on the state-graph.

The key idea of A^* is to introduce a function that *estimates* the distance from a given state to the sink state. By only expanding states for which the sum of the path to that state and the value of the function is minimal, the algorithm does not generate states that are far from the optimal path to the sink state. It was proven by Dechter and Pearl (1985) that if the function never over-estimates the distance to the goal state, this approach finds an *optimal* solution. This is the case with the function that we use in our approach.

To obtain a lower bound on the path from a given state to the goal state, we compute $\frac{1}{2+\alpha}$ times the length of a minimum spanning tree connecting the last location visited in our current state, the locations that are not covered yet, and the depot. In Agatz et al. (2017), we show that in instances where distances fulfill the triangle inequality, a solution $(\mathcal{R}, \mathcal{R})$ to the TSP-D, consisting of a TSP tour $\mathcal{R}$ constructed with the minimum spanning tree heuristic, is a $(2 + \alpha)$-approximation for the TSP-D. Here, α denotes the minimum value for which $\alpha c^d(v, w) \geq c(v, w)$ for every pair of locations $v, w \in V$, as introduced earlier.

3.5 Restricting the number of operations

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 maximum number of truck-only nodes per operation to k. This helps reduce the amount of required work in all passes of the algorithm.

In the small example given in Figure 3, the number of additional arcs that are introduced when we increase k from 0 to 1 and 2 is quite small. Solid arcs are included when $k \geq 0$, dashed arcs are included when $k \geq 1$ and the dotted arc is included when $k \geq 2$. For larger instances, however, the number of introduced arcs increases rapidly. This can be observed from Table 2, where we can see that the number of arcs for an instance with 9 locations and $k = 2$ is already larger than the number of arcs for an instance with 10

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 ;
```

Table 2: The number of arcs in the state-graph when for different values of n and different values of k

	Number of arcs								
n	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$
$n=3$	20	21	21	21	21	21	21	21	21
$n=4$	138	159	160	160	160	160	160	160	160
$n=5$	672	912	944	945	945	945	945	945	945
$n=6$	2630	4200	4670	4715	4716	4716	4716	4716	4716
$n=7$	8976	16566	20206	21016	21076	21077	21077	21077	21077
$n=8$	27986	58625	78820	86065	87346	87423	87424	87424	87424
$n=9$	81920	191904	283744	329104	342096	344000	344096	344097	344097
$n=10$	228942	592722	959214	1188660	1279254	1300842	1303542	1303659	1303660

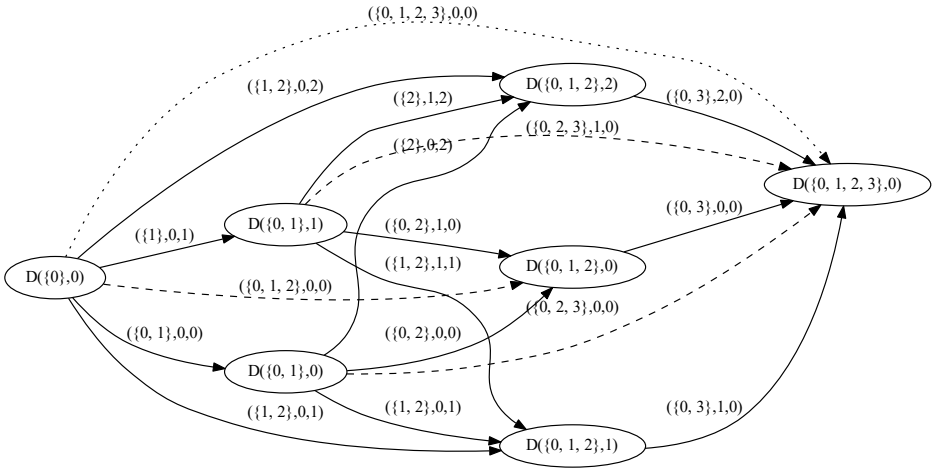


Figure 3: Small part of the state-graph for an instance with four locations. The arcs correspond to operations. The dotted arcs are only included the state-graph when $k \geq 2$, the dashed arcs are only included when $k \geq 1$ and the solid arcs are always included.

locations and $k = 0$.

It is relatively straightforward to adopt our approaches 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 $|S| \leq k + 1$, as drone nodes are added in the second phase and we have by convention $v \notin S$ and $w \in S$. 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. This implies that every sub-problem can be solved in $O(n^{k+1})$ time and thus all solutions can be computed in $O(2^n \cdot n^{k+2})$ time, rather than in $O(3^n n^2)$ time. If we for example forbid truck-only nodes and thus have $k = 0$, we get an $O(2^n \cdot n^3)$ algorithm.

Since A^* often performs best when the branching factor is small, we expect this approach to be most advantageous when we restrict the number of nodes in an operation (small k), given the growth behavior of the number of arcs observed in Table 2. That is, while both the DP-algorithm and A^* benefit from the fact that fewer arcs are evaluated when k is small, the A^* algorithm additionally can take advantage of the fact that fewer

states are expanded in each iteration when k is small.

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

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

We can see that this bound is tight from the example in Figure 4, 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 either 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 first case the round trip time from the depot to either one of the locations is 2 and both locations are visited sequentially. In the second case, both vehicle travel to one location together in one time unit. At that location, the drone travels from the location to the other location in two time units, after which both vehicle travel to the depot in parallel during the fourth and final time unit. In the numerical experiments in Section 4, we evaluate the impact of these restrictions in more practical instances than the presented example.

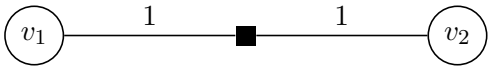


Figure 4: Two customers (circles) that need to be served from the depot (rectangle). Assuming that the truck and drone are equally fast, the optimal solution in case no truck-only nodes are allowed has twice the costs of a solution without restrictions.

4 Numerical experiments

Using the instances presented in Agatz et al. (2017) and several new ones generated using the same approach, we conduct various experiments. The first set of instances consist of points uniformly distributed in a 100×100 two dimensional square with the depot at one of the corners, while the *1-center* and *2-center* instances have locations that are clustered around one and two centers, respectively 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 bound for the A* algorithm.

With these instances, we assess our two implementations of the exact dynamic programming approach: the standard dynamic programming approach (DP) and the A*. Moreover, we also evaluate the 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 $k = \infty$. Unless stated otherwise, we 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. We compare the solution times of our approaches to the time needed to solve the integer linear program (IP) from (Agatz et al., 2017). The IP was solved using CPLEX 12.6.3. Minimum spanning trees

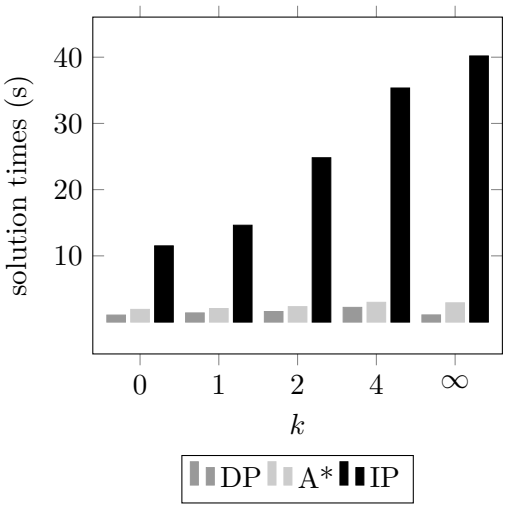


Figure 5: Solution times for different approaches and different number of allowed truck nodes per operation k. Averages over 10 instances with 10 nodes.

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 two different variants of the dynamic programming approach DP and A* for different restrictions on the number of truck nodes k. As an additional benchmark, we also compare the results from the IP as presented in Agatz et al. (2017). This IP uses binary decision variables for each feasible drone operation and constraints that ensure that the associated TSP-D tour 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, many operations are considered and thus fewer variables need to be added to the model. Unfortunately, the number of sub-tour elimination constraints does not decrease when the number of truck nodes is restricted as they are generated for every subset V, independent of the number of operations.

4.1 Comparing running times

Figure 5 provides the running times of the different algorithms for the uniform instances with ten nodes from Agatz et al. (2017). We see that the dynamic programming approaches consistently outperform 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 dynamic programming approach, 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 times of the dynamic programming approaches, we see that the DP is faster than the A^*. This suggests 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 3 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 ten 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 $k = 4$.

4.2 Impact of restrictions

Table 4 shows the impact of the restrictions on the solution quality for the uniform instances. Therefore, we compare the optimal solutions of the approaches that allow at most k truck nodes per 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^∞

k	algo	10	11	12	13	14	15	16	17	18	19	20
0	DP	1	1	2	5	14	37	1:36	4:15	11:02	28:40	1:12:36
0	A*	3	3	6	7	11	29	1:15	1:35	4:59	10:39	27:29
1	DP	2	2	5	15	44	2:12	6:22	18:23	52:21	2:26:32	6:42:34
1	A*	2	5	8	16	27	1:32	4:21	6:28	21:28	55:34	2:08:04
2	DP	1	4	10	34	1:50	6:12	20:12	1:04:54	3:23:13	10:21:06	*
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	2	7	22	1.34	6:57	31:33	2:17:29	9:45:32	*	*	*
4	A*	4	8	19	1:14	3:47	22:05	1:33:51	4:10:13	10:13:05	*	*
∞	DP	1	4	12	56	5:06	26:08	2:38:28	*	*	*	*
∞	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 3: 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

- 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 ($k = 0$). In this case, the average optimality gap ranges from 2.4 percent to 5.9 percent with a maximum gap of 11 percent. While this is a substantial gap, it is much better than the theoretical worse-case gap of 50 percent from Theorem 3.5. 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 with less truck nodes.

Next, we consider the impact of the restrictions on the solution quality for different relative drones speeds (α) and different types of instances. Figure 6 shows that limiting k has a larger negative impact at lower drone speeds. Especially the difference between

	$k=0$			$k=1$			$k=2$			$k=4$		
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 4: Solution quality for different numbers of allowed truck nodes per operation k, uniformly distributed, $\alpha = 2$, averages over ten instances

$k = 0$ and $k = 1$ is substantial. This is intuitive as it is more beneficial for the truck to visit additional nodes separated from the drone when the drone is relatively slow. This suggests that the restrictions are most suitable in situations where the drone is faster than the truck.

Figure 7 shows a similar trend as the performance at lower k-values is worse for the clustered instances than the uniform instances. In clustered instances, it is beneficial to have the truck serve the clustered nodes in the centers while the drone visits the outliers. However, this is not possible when we restrict the number of truck nodes in an operation.

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 8. 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 in the case with restrictions. As noted before, the figure shows that this effect is more pronounced in the clustered instances, i.e., 1-center and 2-center. In line with this observation, we see relative more drone nodes in the uniform instances than the clustered instances in the unrestricted cases.

Figure 9 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, without restrictions, the drone generally incurs more waiting time than the truck. This is related to the fact that our drone is faster than the truck. As expected, we see that the waiting time of the truck decreases with the maximum number of allowed truck nodes per operation. When comparing the different instance types, we see that the waiting times for the drone are

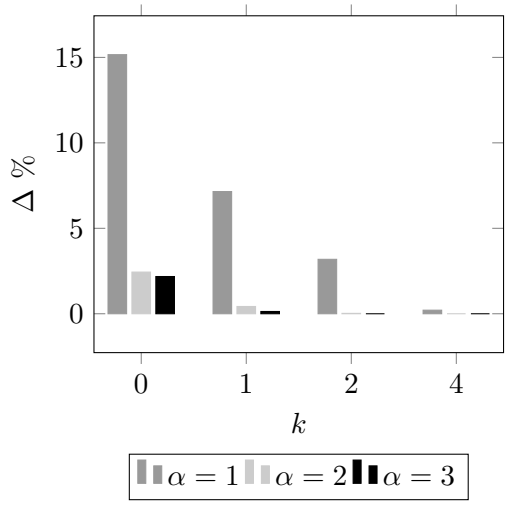


Figure 6: Solution quality for different drone speeds α and different numbers of allowed truck nodes per operation k. Averages over 10 uniform instances with 10 nodes.

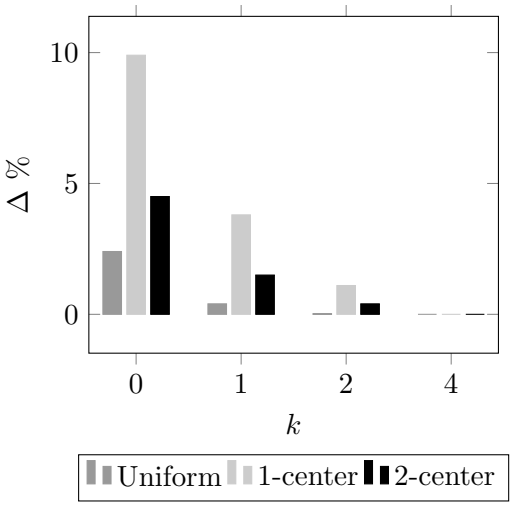
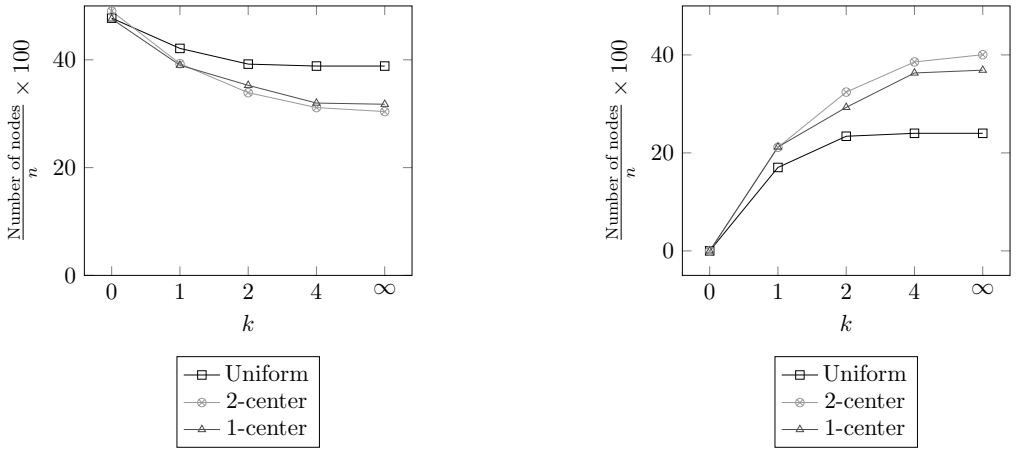


Figure 7: Solution quality for different instance types and different numbers of allowed truck nodes per operation k. Averages over 10 instances with 10 nodes.



(a) Number of drone nodes

(b) Number of truck nodes

Figure 8: Number of drone and truck nodes in the optimal solutions for different maximum numbers of truck nodes per operation k for different types of instances, averages over 60 instances of size (n) 10 to 15

longer in the uniform instances than the clustered instances. A potential reason for this is that the drone makes shorter flights in the uniform case which together with its higher speed leads to more waiting.

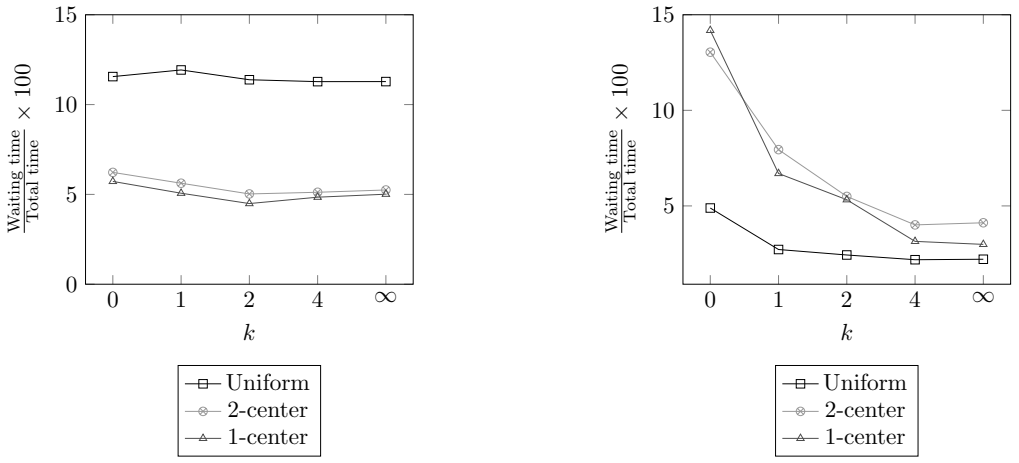
5 Conclusions and future research

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 reduce the running times of our algorithm, future research could investigate and develop extensions of this approach. 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.

Furthermore, it would be interesting to investigate to what extent the techniques developed for the TSP-D could be extended for a problem version in which drones can depart



(a) Drone waiting time

(b) Truck waiting time

Figure 9: 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

an *any* part of the route, or at specific locations (which do not need to be restricted to customer locations).

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/`.

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.

K.G Orphanides. Drones can now land on moving cars. *Wired*, 2016. URL `http://www.wired.co.uk/article/drone-lands-on-moving-car`.

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. Master's thesis, University of Padova, Italy, 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.