Parameterized Algorithms for the Drone Delivery Problem

Simon Bartlmae
University of Copenhagen, Denmark
Institute of Computer Science, University of Bonn, Germany

Andreas Hene
Institute of Computer Science, University of Bonn, Germany

Joshua Könen
Institute of Computer Science, University of Bonn, Germany

Heiko Röglin
Institute of Computer Science, University of Bonn, Germany

Abstract

Timely delivery and optimal routing remain fundamental challenges in the modern logistics industry. Building on prior work that considers single-package delivery across networks using multiple types of collaborative agents with restricted movement areas (e.g., drones or trucks), we examine the complexity of the problem under structural and operational constraints. Our focus is on minimizing total delivery time by coordinating agents that differ in speed and movement range across a graph. This problem formulation aligns with the recently proposed *Drone Delivery Problem with respect to delivery time* (DDT), introduced by Erlebach et al. [ISAAC 2022].

We first resolve an open question posed by Erlebach et al. [ISAAC 2022] by showing that even when the delivery network is a path graph, DDT admits no polynomial-time approximation within any polynomially encodable factor $a(n)$, unless P=NP. Additionally, we identify the *intersection graph* of the agents, where nodes represent agents and edges indicate an overlap of the movement areas of two agents, as an important structural concept. For path graphs, we show that DDT becomes tractable when parameterized by the treewidth w of the intersection graph, and we present an exact FPT algorithm with running time $f(w) \cdot \text{poly}(n, k)$, for some computable function f. For general graphs, we give an FPT algorithm with running time $f(\Delta, w) \cdot \text{poly}(n, k)$, where Δ is the maximum degree of the intersection graph. In the special case where the intersection graph is a tree, we provide a simple polynomial-time algorithm.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Theory of computation → Graph algorithms analysis

Keywords and phrases Complexity, Delivery, FPT algorithms, Graph Theory

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.8

1 Introduction

Drone-based package delivery has the potential to transform the last-mile segment. Recent studies foretell that utilizing unmanned aerial vehicles (i.e., drones) can have a great sustainability impact while meeting increasing demands of customers [13, 17]. Additionally, it is observed that in the U.S. approximately 10–15% of package deliveries are delayed due to increasing loads [7]. According to ElSayed et al. [10], recent surveys indicate that a vast majority of people prefer a same-day delivery, which proves challenging for the logistics industry. Consequently, large companies focus more and more on developing suitable logistics models incorporating drones. Apart from monetary benefits, the utilization of drones can be very helpful in supplying medicine or food in areas struck by war or natural disasters [2].

In this work, we study a cooperative delivery setting recently introduced by Erlebach et al. [11] in which drones can hand over packages and work together to achieve the best possible delivery schedule. Each agent operates within a restricted movement area on the graph and is assumed to have unlimited battery capacity. In general, this model is inspired by real-world constraints such as restricted air spaces or environmental circumstances. Restricted movement areas also capture settings with different kinds of agents, not necessarily only drones – different types of agents might be eligible to traverse certain parts of a graph while others might not. Package handover between drones is already implemented in industry; companies such as IBM have developed and patented parcel transfer methods since 2017 [21].

Our focus is on *minimizing total delivery time*, a problem referred to as the *Drone Delivery Problem with respect to time* (DDT) [3, 11]. We study general graphs as well as the special case of path graphs. For path graphs, we first establish a hardness result, followed by the first fixed-parameter tractable algorithms for the path as well as the general settings. For some parameter k, an algorithm is fixed-parameter tractable (FPT) if it runs in time $f(k) \cdot \text{poly}(n)$, where f is a computable function; such algorithms are efficient for small parameters and useful for fine-grained complexity analysis. We analyze the complexity of DDT using the *intersection graph* of the agents, where nodes represent agents and edges indicate overlapping movement areas. The *treewidth* and maximum degree Δ of this graph serve as natural parameters for characterizing instance complexity – especially since DDT is already NP-hard on path graphs. These parameters reflect the extent of agent overlap and potential interference, and are likely to be small in practical settings.

Related Work. For the most part, research regarding collaborative drone delivery focuses on two main objectives: minimizing fuel consumption and minimizing delivery time. In this work, we focus exclusively on the latter. Research on minimizing total consumption differs fundamentally from our analysis, since in our setting there are no battery constraints; therefore, the results do not translate. For work on minimizing fuel consumption, we refer to [5, 11]. Regarding delivery time, Bärtschi et al. [4] and Carvalho et al. [8] show that the problem of delivering a single package can be solved in polynomial time if the agents are free to move without restrictions – even with predetermined starting positions. However, Carvalho et al. [8] also prove that the problem becomes NP-hard if there are at least two packages involved. For the case of restricted movement areas, Erlebach et al. [11] prove that it is NP-hard to approximate DDT within a factor of $O(\min\{n^{1-\varepsilon}, k^{1-\varepsilon}\})$, even if agents have equal speeds and fixed starting positions. Here, n denotes the number of vertices in the graph and k the number of agents. In the problem variant in which the starting positions are not given but can be chosen by the algorithm, they show that no polynomial-time approximation algorithm with any finite approximation ratio exists, unless P=NP. They also show that DDT with fixed starting positions remains NP-hard on path graphs if agents can have arbitrary speeds.

Bartlmae et al. [3] recently refined these results, showing that the problem on a path is still NP-hard if only two different speeds are allowed. This closes the gap regarding speeds as these instances can be solved trivially for only one speed. Additionally, they show that for the special case of unit grid graphs and only two different speeds, the problem remains NP-hard for fixed starting positions and is hard to approximate within a factor of $O(n^{1-\varepsilon})$ for selectable starting positions, as well as agents constrained to movement areas forming rectangles. They also provide a simple n-approximation algorithm for this setting. Complementing these hardness results, Luo et al. [16] demonstrated that on a path with fixed starting positions and exactly two speeds the problem can be solved in polynomial time; the complexity for three or more speeds, however, remains open.

Our Results. In Section 3.1, we provide a hardness result for DDT on path graphs with selectable starting positions, closing open questions posed by Erlebach et al. [11], Bartlmae et al. [3] and Luo et al. [16].

▶ **Theorem 1.** *DDT on a path with selectable starting positions is $a(n)$-APX-hard for any function $a : \mathbb{N} \to \mathbb{R}_+$ that can be computed in polynomial time.*

We then present the first FPT algorithm for this setting in Section 3.2, parameterized by the treewidth of the agents' intersection graph.

▶ **Theorem 2.** *DDT on a path with selectable starting positions can be solved in time $O(2^w \cdot w^2 \cdot k)$, where w denotes the treewidth of the intersection graph.*

In Section 4, we consider the more complex case of general graphs. We design an FPT algorithm whose running time depends on the treewidth and maximum degree of the intersection graph.

▶ **Theorem 3.** *DDT with selectable starting positions can be solved in time $f(\Delta, w) \cdot \text{poly}(n, k)$ on general graphs, where w denotes the treewidth and Δ denotes the maximum degree of the intersection graph and f is a function in w and Δ.*

For the special case where the intersection graph is a tree, we show in Section 4.3 that DDT can be solved in polynomial time. This result is not implied by the previous theorems; it allows for arbitrary maximum degree in the intersection graph.

▶ **Theorem 4.** *DDT on a general graph with selectable starting positions can be solved in time $O(k^2 \cdot n^2)$ if the underlying graph of the intersection graph is a tree.*

2 Preliminaries

In this section, we formally define the Drone Delivery Problem with respect to time (DDT).

Setting. We define an instance of DDT as a tuple $I = (G, (s, t), A)$, where $G = (V, E, \ell)$ with $n = |V|$, is an undirected graph with edge lengths $\ell : E \to \mathbb{R}_{\geq 0}$. The package starting position, as well as its destination, are given by (s, t), where $s \in V$ and $t \in V$. The set A represents all agents, with $k = |A|$. There are two settings regarding agents which are typically considered for DDT:

1. DDT with **selectable positions** (*DDT-SP*): Each agent $a \in A$ is defined as a tuple $a = (v_a, G_a)$, where v_a is the agent's speed and $G_a = (V_a, E_a)$ is a connected subgraph of G representing the agent's movement area. The initial starting position of a can be chosen freely once before delivery is initiated.
2. DDT with **fixed positions** (*DDT-FP*): Each agent $a \in A$ is a tuple $a = (v_a, G_a, p_a)$, with v_a and G_a defined as above, and the additional parameter $p_a \in V_a$ specifies the fixed initial starting position of the agent.

An agent a traverses an edge $\{u, v\} \in E_a$ in time $\frac{\ell(\{u,v\})}{v_a}$, regardless of whether it is carrying the package or not. Whenever two agents meet at the same vertex, they can hand over the package instantaneously. We define the travel time of agent a between two vertices u, v as the length of the shortest path in G_a divided by v_a and denote it by $d_a(u, v)$. If either u or v is not contained in V_a, we define $d_a(u, v) = \infty$. For any pair of vertices and any possible agent the values can be precomputed in polynomial time.

We specify a feasible solution to be a schedule S of consecutive trips by agents delivering the package from s to t. More precisely we want the schedule to consist of two components: A complete edge-by-edge trip of every agent and a complete edge-by-edge trip for the package

specifying the current package carrier at any point in time. For the first part we want for every agent a a set of sorted triples $\mathcal{T}_a$. Every triple (u, v, τ) indicates that agent a moves over edge $\{u, v\}$ starting at time τ. It has to hold that $\{u, v\} \in E_a$ and that for two consecutive triples (u, v, τ) and (u', v', τ') that $u' = v$ and $\tau + \frac{\ell(\{u,v\})}{v_a} \leq \tau'$. Moreover, for DDT-SP we need to provide the starting position p_a for every agent $a \in A$ and for DDT-FP we need to have for the first triple that $u = p_a$. The trip of the package is defined as a sorted set of tuples $\mathcal{T}$ of the form (u, v, a, τ), indicating that agent a carries the package over edge $\{u, v\}$ starting at time τ. A schedule is feasible only if, for every tuple $(u, v, a, \tau) \in \mathcal{T}$, it holds that $(u, v, \tau) \in \mathcal{T}_a$. Again for two consecutive tuples (u, v, a, τ) and (u', v', a', τ') we also require $u' = v$ and $\tau + \frac{\ell(\{u,v\})}{v_a} \leq \tau'$. Additionally, the first tuple must satisfy $u = s$, and the last tuple must satisfy $v = t$. The total delivery time is defined as $c(S) := \tau + \frac{\ell(\{u,v\})}{v_a}$, assuming (u, v, a, τ) is the final tuple in the schedule.

An instance of DDT-SP is illustrated in Figure 1a. Throughout this paper we will focus exclusively on DDT-SP.

Useful Properties. There are several useful properties regarding DDT. Erlebach et al. [11] demonstrated that there exists an optimal schedule in which every involved agent picks up and drops off the package exactly once. The key insight is that instead of using an agent multiple times we let that agent carry the package until its final drop-off without increasing the delivery time.

Another important observation relates to trips made by agents:

▶ **Observation 5.** *In DDT-SP there exists an optimal schedule such that all involved agents move if and only if they carry the package.*

Placing agents initially at the first vertex of their respective trip immediately leads to this observation. Unless stated otherwise, this implicit initial positioning is assumed.

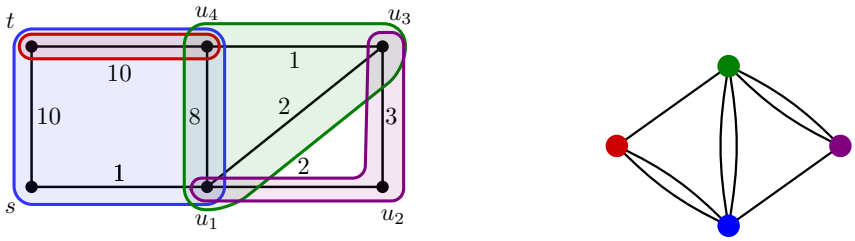
Intersection Graph. A useful structural concept is the *intersection graph* of the agents. We define the intersection (multi)graph G_I of a DDT instance $(G, (s, t), A)$ as follows:

- The vertex set $V(G_I)$ is the set of agents A.
- For any two agents $a, a' \in A$ with corresponding vertex sets V_a and $V_{a'}$ in G, we add $|V_a \cap V_{a'}|$ parallel edges between a and a' in G_I – one for each vertex they share in G. Formally, the edge set $E(G_I)$ is defined as the multiset

$$E(G_I) = \biguplus_{\substack{a,a' \in A \\ a \neq a'}} \{ \underbrace{\{a, a'\}, \ldots, \{a, a'\}}_{|V_a \cap V_{a'}| \text{ times}} \}.$$

Throughout this paper, we treat E as a multiset and omit explicit edge indices where they are not relevant. Intuitively, a feasible schedule corresponds to a path in the intersection graph that starts with an agent intersecting s and ends with an agent intersecting t. If we assume each agent is used at most once, this path will be simple. Figure 1b illustrates the intersection graph for a given instance. We denote by $\Delta(G_I)$ the maximum degree of the intersection graph G_I. When it is clear from the context, we will write Δ instead of $\Delta(G_I)$. For any vertex u, we define B_u as the set of agents that cover u, i.e., $B_u := \{a \in A \mid u \in V_a\}$.

Tree Decomposition. For computing the optimal traversal of the agents, we will use the concept of a *tree decomposition*. A tree decomposition is used as a measure of how similar a given graph is to a tree and was first introduced by Robertson and Seymour [18]. Computing the optimal treewidth is NP-hard [1], but finding the optimal tree decomposition is fixed-parameter tractable if parameterized by treewidth [6].



(a) (b)

Figure 1 Fig. 1a shows a DDT instance with four agents, each restricted to a colored movement area (bounding boxes). The red agent has speed 10, the blue and green agents have speed 1, and the purple agent has speed 2. In the optimal schedule, the blue agent starts at s and delivers the package to u_1 in time 1. The green agent then transports it via u_3 to u_4 in time $2 + 1 = 3$. Finally, the red agent delivers it to the target t in time 1, for a total delivery time of 5. Fig. 1b shows the corresponding intersection graph.

▶ **Definition 6** (Tree Decomposition). *A tree decomposition of a graph $G = (V, E)$ is a pair $\mathcal{T} = (\mathbb{T}, \{X_t\}_{t \in V(\mathbb{T})})$ where $\mathbb{T}$ is a tree with root $r \in V(\mathbb{T})$ and every node $t \in V(\mathbb{T})$ is associated with a set of the vertices $X_t \subseteq V$, called bag. Additionally, the following three conditions must be fulfilled:*

- $\bigcup_{t \in V(\mathbb{T})} X_t = V(G)$, *i.e., every vertex $v \in V(G)$ appears in at least one bag,*
- *for every edge $\{u, v\} \in E(G)$ there must exist a bag $t \in V(\mathbb{T})$ such that the vertices u and v are contained in this bag,*
- *the set of nodes $T_u = \{t \in V(\mathbb{T}) \mid u \in X_t\}$ forms a connected subtree of $\mathbb{T}$ for every choice $u \in V(G)$.*

The *width* of a tree decomposition is the size of its largest bag minus one. The *treewidth* of a graph G is the minimum width over all possible tree decompositions of G, denoted by w. To simplify the algorithm design, we assume the decomposition has a specific structure known as a *nice tree decomposition*: a binary tree with root r and leaves l such that $X_r = \emptyset$ and $X_l = \emptyset$ for all $l \in V(\mathbb{T})$, containing only three types of nodes:

- **Introduce node**: For two nodes $t_{\text{parent}}, t_{\text{child}} \in V(\mathbb{T})$ where t_{parent} has exactly one child t_{child}, t_{parent} is an *introduce node* iff $X_{t_{\text{parent}}} = X_{t_{\text{child}}} \cup \{v\}$ for some $v \in V(G)$.
- **Forget node**: For two nodes $t_{\text{parent}}, t_{\text{child}} \in V(\mathbb{T})$ where t_{parent} has exactly one child t_{child}, t_{parent} is a *forget node* iff $X_{t_{\text{parent}}} \cup \{v\} = X_{t_{\text{child}}}$ for some $v \in V(G)$.
- **Join node**: For three nodes $t_{\text{parent}}, t^1_{\text{child}}, t^2_{\text{child}} \in V(\mathbb{T})$ where t_{parent} has exactly two children t^1_{child} and t^2_{child}, t_{parent} is a *join node* iff $X_{t_{\text{parent}}} = X_{t^1_{\text{child}}} = X_{t^2_{\text{child}}}$.

As in [9] we additionally assume that the edges are also introduced like the vertices in their respective type of node:

- **Introduce edge node**: For every edge $e = \{u, v\} \in E$ there is exactly one introduce edge node $t \in V(\mathbb{T})$ with $u, v \in X_t = X_{t'}$ for its only child bag t' and $X_t = X_{t'}$.

We assume that for every edge $\{u, v\}$ the corresponding introduce edge node appears directly before either u or v is forgotten in an ancestor node.

Using all node types, we define V_t and E_t as the set of vertices and edges that are either contained or introduced in bag t or were introduced in some child of t. Additionally, we define $G_t = (V_t, E_t)$ as the subgraph revealed up to node t. As a consequence for the construction of the introduce edge nodes, for every join node $t \in V(\mathbb{T})$ there are no edges with both endpoints in X_t, i.e. $E_t \cap \{\{u, v\} \mid u, v \in X_t\} = \emptyset$.

Based on a tree decomposition of width w for some graph G, we can always compute a nice tree decomposition (with or without introduce edge nodes) of width w with $O(|V(G)| \cdot w)$ nodes in polynomial time [9].

3 Drone Delivery on Path Graphs

We consider DDT-SP instances where the underlying graph is a path: a sequence of vertices connected in a line, with degree two at each internal vertex and degree one at the two endpoints. For an agent a, we denote its movement area by $[s_a, f_a]$, where s_a and f_a are the leftmost and rightmost vertices that a can access. This setting can also be interpreted as agents moving along a one-dimensional line, with each movement area corresponding to an interval on that line, as illustrated in Figure 2. This allows us to interpret DDT-SP on a path as an instance of the *subinterval covering* problem introduced by Luo et al. [16], where the objective is to select, for each agent's interval, a subinterval such that the union covers $[s, t]$, while minimizing the total cost defined by the length of each selected subinterval divided by the agent's speed.

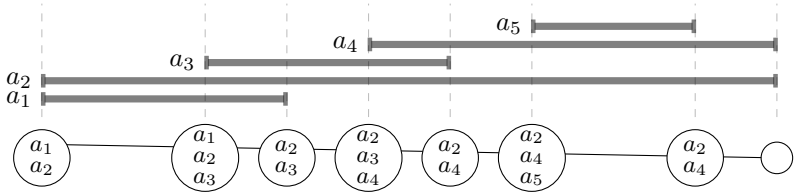


Figure 2 A DDT-SP instance on a line and a corresponding tree decomposition of its intersection graph. At every start or end position we either add or remove the corresponding agent.

For the setting with fixed initial positions, NP-hardness has been proven [3, 11], although strong NP-hardness remains unresolved. In contrast, for the setting with selectable positions – which is the primary focus of this paper – Luo et al. [16] presented a polynomial time algorithm for the case of exactly two speeds; for three or more speeds, however, no results were known prior to our work.

Before presenting parameterized algorithms for DDT on path graphs, we first establish hardness results for this setting, which motivates the relevance and necessity of parameterized approaches.

3.1 Inapproximability

Instead of merely proving NP-hardness, we establish a much stronger result: no polynomial-time algorithm with any polynomially encodable approximation ratio exists, unless P=NP.

▶ **Theorem 1.** *DDT on a path with selectable starting positions is $a(n)$-APX-hard for any function $a : \mathbb{N} \to \mathbb{R}_+$ that can be computed in polynomial time.*

In the following we only present a high-level overview, the complete proof can be found in the full version of the paper. To prove the inapproximability, we reduce from the PartitionInto-k problem.

▶ **Definition 7** (PartitionInto-k)**.** *Given a set of natural numbers $p_1, \ldots, p_n$ and an integer k, the task is to decide whether the index set $\{1, \ldots, n\}$ can be partitioned into k disjoint subsets $I_1 \dot\cup \ldots \dot\cup I_k$ such that $\sum_{i \in I_1} p_i = \cdots = \sum_{i \in I_k} p_i$.*

Note that in this subsection k refers to the number of subsets (and not the agent set). The NP-hardness of PartitionInto-k follows immediately from the hardness of Partition with $k = 2$; however for the proof of Theorem 1 we require the stronger result that PartitionInto-k is also NP-hard in the strong sense, which is included in the full version.

▶ **Theorem 8.** PartitionInto-k *is NP-hard in the strong sense (even with distinct integers).*

This is done by a straightforward reduction from 3-Partition, which was originally shown to be strongly NP-hard by Garey and Johnson [12] and for distinct integers by Hullet et al. [14]. It is particularly important that k is part of the input; for a fixed k, the problem admits an FPTAS, as shown by Sahni [20].

To show that no approximation algorithm $\mathcal{A}$ with a polynomially encodable approximation ratio $a(n)$ can exist (unless P=NP), we demonstrate that such an algorithm could be used to solve PartitionInto-k in polynomial time. We do so by constructing a DDT instance I' from a given PartitionInto-k instance I such that:

I is a "yes"-instance of PartitionInto-k if and only if $c_{\mathcal{A}}(I') \leq d \cdot a(n)$, where $c_{\mathcal{A}}(I')$ is the delivery time of the schedule produced by $\mathcal{A}$ for the instance I' and the value d is determined by the construction.

Let $I = (p_1, \ldots, p_n)$ be an arbitrary PartitionInto-k instance with distinct integers. We assume that the numbers are sorted in decreasing order, i.e., $p_1 > \cdots > p_n$. We denote the total sum of all elements by P.

For each object p_i we create k drones – one for each partition set. We refer to these drones as *element agents* and denote them by $e_{i,j}$, where $e_{i,j}$ is associated with object p_i and partition set $j \in \{1, \ldots, k\}$. The speed of a drone $e_{i,j}$ is denoted by v_j, meaning that all drones associated with the same partition set (i.e., with the same index j) share the same speed. We place several gadgets along the line and aim to ensure that each element agent can only be placed and transport the package at specific designated positions and not at arbitrary locations along the path. To achieve this, we employ two key mechanisms: first, we restrict the movement areas of the agents such that they are clearly unable to assist outside their interval. Second, we choose the ratios between the speeds $v_1, \ldots, v_k$ to be sufficiently large, allowing us to define long intervals that certain agents cannot traverse in their entirety without exceeding the time bound $d \cdot a(n)$.

For each of the k partition sets, we create a *partition gadget*, that enforces that the sum of the weights p_i associated with the element agents $e_{i,j}$ placed at that gadget is at least $\frac{P}{k}$. Placing an element agent $e_{i,j}$ at a partition gadget can thus be interpreted as assigning p_i to the corresponding partition set.

Additionally, for each object p_i we construct a gadget O_i – called an *object gadget* – that ensures each object is assigned to at most one partition set. This is done by requiring that exactly $k - 1$ of the agents $e_{i,1}, \ldots, e_{i,k}$ must be present at the object gadget, resulting in at most one of them being placed at a partition gadget. As a result, we can ensure that each partition gadget ends up with a total weight of exactly $\frac{P}{k}$, if such a partition exists. If not, the construction forces a very large delivery time, causing the bound $d \cdot a(n)$ to be exceeded.

An example layout of the gadgets, ordered as $P_1, \ldots, P_k, O_n, \ldots, O_1$, is shown in Figure 3. Consecutive gadgets are placed sufficiently far apart so that only specially designated fast transition agents can traverse the gaps, ensuring that each element agent can contribute to at most one gadget.

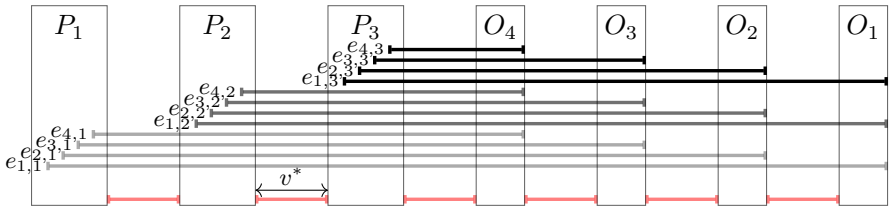


Figure 3 Overview of all gadgets in an example with four objects and three partition sets. The element agents are shown in gray, with darker shades indicating higher speeds. Transition agents (marked in red) with speed $v^* \gg v_1, \ldots, v_k$ are placed between gadgets; if these are bypassed and an element agent is used at multiple gadgets, the time bound $d \cdot a(n)$ is exceeded.

3.2 A Parameterized Algorithm

We now design an FPT algorithm for DDT-SP on path graphs. As a parameter for our approach, we use the maximum overlap – the *thickness* $\sigma := \max_{u \in V(G)} |B_u|$, i.e., the largest number of agents covering any single vertex.

Since each agent corresponds to an interval on the line, the intersection graph of agents is an interval graph. In such graphs, agents covering a common point form a clique, and each maximal clique corresponds to some point on the line [15]. As interval graphs are chordal, their treewidth equals the size of the largest clique minus one [19], which in our case implies $w = \sigma - 1$. Thus, the treewidth of the intersection graph and the thickness parameter are equivalent. This structural property allows for a dynamic programming algorithm over a tree decomposition of the intersection graph. We define the *event points* $e_1 < \cdots < e_{2k}$ as the start and end positions of all agent intervals, i.e., the multiset $\{s_a, f_a \mid a \in A\}$. We sort these events from left to right along the path $(e_1 < \cdots < e_{2k})$, resolving ties by placing start points before end points; the order among start points or among end points is chosen arbitrarily. In an optimal schedule, handovers occur only at these event points – specifically at $s_{a'}$ or f_a for a handover from a to a' – since otherwise extending the faster agent's segment would yield a strictly better or equally good schedule.

We interpret each event e_i as a node in a tree decomposition that forms a path, see Fig. 2. We can consider the set of agents covering e_i, Z_{e_i} as the bag at position i. If e_i is the start vertex of some agent a, we say that a is *introduced* to the bag, and if e_i is the end point of some agent a, it is *forgotten*. We define a dynamic program over the path decomposition induced by this event sequence. At each e_i, we maintain:

- the *bag* Z_{e_i} of agents covering e_i,
- a subset $S \subseteq Z_{e_i}$ of agents already used,
- a current carrier $a \in Z_{e_i}$ responsible for transporting the package from e_i to e_{i+1}.

Since at most σ agents cover any point, we have $|Z_{e_i}| \leq \sigma$, and the number of subsets $S \subseteq Z_{e_i}$ (representing already-used agents) is bounded by 2^σ. The dynamic programming table stores the minimum delivery time to reach e_i under the assumption that agent a will deliver it to e_{i+1} and the current set of already used agents is S, denoted by $D[e_i, S, a]$. We assume no agent starts at t or ends at s, as those movement intervals can be cut off without affecting the solution. Furthermore, at most one handover occurs per event point, and it happens only if $e_i = s_{a'}$ or $e_i = f_a$, as argued before. While we only store delivery times in the DP table, the full delivery sequence can be reconstructed via backtracking.

In the following we always assume for some DP entry $D[e_i, S, a]$ that $a \in S$, otherwise the entry is defined as ∞. Assume we have already computed all possible entries $D[e_{i-1}, S \subseteq Z_{e_{i-1}}, a \in Z_{e_{i-1}}]$ correctly, and now wish to compute $D[e_i, S \subseteq Z_{e_i}, a \in Z_{e_i}]$.

The formal correctness of the following lemmas appears in the full version of the paper.

Introduce Event. For an introduce event e_i, it is the start point of an agent a', so a' is now newly contained in Z_{e_i}. We differentiate between a' being included into S, which corresponds to the package being handed over to a' and the case where a' is omitted (for now). Note that when a' enters S it will also be the dedicated package carrier.

▶ **Lemma 9.** *Let e_i be an introduce event that introduces some agent a'. Assume that for all valid combinations the solutions $D[e_{i-1}, S' \subseteq Z_{e_{i-1}}, a \in S']$ have been computed correctly, then we correctly compute $D[e_i, S \subseteq Z_{e_i}, a \in S]$ in time $O(\sigma)$.*

Forget Event. For a forget event e_i, an end point for some agent a' was reached, thus a' is removed from Z_{e_i}. We consider all possibilities for how a' could have been involved in an optimal delivery: Either a' delivered the package to e_i and now hands it over to a different agent a, or a' was involved in the optimal delivery but did not carry the package to e_i, or a' never held the package at any point. Out of all of these options we again take the solution variant with the smallest cost.

▶ **Lemma 10.** *Let e_i be a forget event that forgets some agent a'. Assume that for all valid combinations the solutions $D[e_{i-1}, S' \subseteq Z_{e_{i-1}}, a \in S']$ have been computed correctly, then we correctly compute $D[e_i, S \subseteq Z_{e_i}, a \in S]$ in time $O(1)$.*

▶ **Theorem 2.** *DDT on a path with selectable starting positions can be solved in time $O(2^w \cdot w^2 \cdot k)$, where w denotes the treewidth of the intersection graph.*

Proof. The correctness follows from the correctness of the different types of events and that the optimal solution must be stored in some entry $D[e_j, S \subseteq Z_{e_j}, a]$ for some final end point event $e_j = e_{2k}$. Since for every type of event we have runtime at most $O(\sigma)$ and the set of instances at an event point e_i is upper bounded by $2^\sigma \cdot \sigma$ and $2k$ event points exist, we get a final runtime of $O(2^\sigma \cdot \sigma^2 \cdot k) = O(2^w \cdot w^2 \cdot k)$. ◀

4 Fixed-Parameter Tractability on General Graphs

Before designing an FPT algorithm on general graphs, we argue that, w.l.o.g., the intersection graph can be assumed to be simple – i.e., agents intersect at most once. Any instance can be transformed accordingly, increasing degree and treewidth by only a constant.

4.1 Unique Intersections between Agents

Given any DDT instance $(G, (s, t), A)$, we now create an instance $(G', (s', t'), A')$ that has equivalent schedules, but its intersection graph G'_I is simple and therefore any pair of agents overlap in at most one vertex. Each agent $a \in A$ originally moves on a (possibly overlapping) subgraph G_a of G. To guarantee that no pair of agents share multiple vertices, we create a *disjoint copy* of every G_a and relabel each vertex u as (u, a). The k original agents are then confined to their respective copies G'_a. Whenever two subgraphs G_a and G_b share a vertex u in G, the transformed graph G' now contains two distinct copies, i.e., (u, a) and (u, b). We connect these copies of the same vertex by edges (of length 0) and introduce a dedicated *auxiliary agent* h_u with speed 1 and movement area restricted to the newly added edges of length 0 between the copies. Thus, the package can be transferred instantly between any two copies of the same original vertex, while original agents share no common vertex. The

resulting instance preserves all feasible schedules with the same delivery time as the original one but it also satisfies the new constraint that any pair of agents shares at most one vertex, as any pair of original agents a', b' does not share a common vertex in G', neither does any pair of helping agents $h_u, h_{u'}$, and agents a' and h_u may only intersect in (u, a). In the full version of the paper, we formally define such a transformation and prove the following lemma.

▶ **Lemma 11.** *For any DDT instance $(G, (s,t), A)$, there exists an equivalent instance $(G', (s', t'), A')$, where $w(G'_I) \leq w(G_I) + 1$, $\Delta(G'_I) \leq \Delta(G_I) + 1$ and G'_I is simple.*

4.2 Tree Decomposition Algorithm for Unique Intersections

Let $(G, (s,t), A)$ be a drone delivery instance in which each pair of agents $a, a' \in A$ intersects in at most one vertex. For an agent $u \in A$, we define $N_u := \{v \in V(G_I) \mid \{u, v\} \in E(G_I)\}$. Let $D : A \times A \to V(G) \cup \{-\}$ be a function that returns for two agents a_i, a_j their unique intersection point $V_i \cap V_j$ if it exists and $-$ otherwise. Let G_I be the intersection graph, and assume we have a nice tree decomposition $\mathcal{T} = (\mathbb{T}, \{X_t\}_{t \in V(\mathbb{T})})$ of G_I of width w and maximum degree $\Delta = \max_{u \in V(G_I)} |N_u|$.

We now design a dynamic program over $\mathcal{T}$ that is parameterized by w and Δ and computes the optimal agent tour. The algorithm is inspired by standard treewidth-based DP techniques for TSP[1]. For each node $t \in V(\mathbb{T})$, we process the subgraph $G_t = (V_t, E_t)$.

A delivery tour can be modeled as an ordered sequence $(a_1, a_2, \ldots, a_b)$ of agents. Each contiguous subsequence $A_{i,j} = (a_i, \ldots, a_j)$ induces a valid subpath in G_t, if the corresponding edges $\{a_l, a_{l+1}\} \in E(G_t)$ for all $l \in [i, j-1]$, and $a_i, a_j \in X_t$. Intermediate agents a_l with $i < l < j$ incur cost $d_{a_l}(D(a_{l-1}, a_l), D(a_l, a_{l+1}))$.

However, to determine the cost of agents a_i, a_j we require knowledge about the corresponding unknown preceding and succeeding agents to determine entry and exit points. Thus, the DP state additionally stores for the boundary agents in X_t their assumed predecessor and successor. This allows consistent cost computation when merging subpaths at join nodes.

We slightly modify the input instance to streamline boundary cases: we add two artificial agents a_s and a_t, each covering only vertex s and t respectively, and having zero speed. The unique intersection property extends naturally to these agents. We now additionally require any valid delivery tour to start with a_s and end with a_t. Both agents are required to be always contained in any bag, increasing the treewidth by exactly 2. Note that the actual delivery can still start at s, since a_s can immediately hand over the package at no cost.

To simplify boundary handling, we also introduce auxiliary agents a'_s and a'_t, each identical to a_s and a_t, but always assumed to lie outside the tree decomposition (i.e., they are never contained in any bag). A delivery tour will now implicitly require the tour to start with a'_s and end with a'_t. Together, this guarantees that for each boundary agent, which always includes a_s and a_t, its predecessor and successor are explicitly known, enabling us to compute exact cost for partial solutions.

Each DP state $D[t, B_t^0, B_t^1, B_t^2, M, \hat{A}, \hat{Z}]$ represents the optimal forest of paths in subgraph G_t that connects agents according to the following:

- $B_t^i \subseteq X_t$ for $i \in \{0, 1, 2\}$ contains all agents with degree exactly i
- $M \subseteq B_t^1 \times B_t^1$ encodes directed pairwise path endpoints as a matching
- $\hat{A}$ and $\hat{Z}$ assign known predecessor/successor agents to agents in B_t^1:
  - $\hat{A}$ maps to a neighbor agent s.t. the corresponding edge is not contained in E_t
  - $\hat{Z}$ maps to the neighbor s.t. the corresponding edge is contained in E_t

[1] See e.g. `http://www.cs.bme.hu/~dmarx/papers/marx-warsaw-fpt2`, page 17.

Each path P_j in a feasible forest F must satisfy the following conditions:

- The endpoints of P_j lie in B_t^1 and every agent $a \in B_t^i$ has degree i in F
- For every pair $(a, a') \in M$, there must be a path P_j in F with both of these as endpoints

The cost of a forest F is then evaluated using the previously mentioned reasoning of computing the cost of every individual agent measured by its two neighboring agents in the corresponding sequence. For convenience we will sometimes represent functions as set of tuples, i.e. $\hat{A} = \{(u, \hat{A}(u)) \mid u \in V(G_t)\}$.

For the root node r the DP entry $D[r, \emptyset, \{a_s, a_t\}, \emptyset, \{(a_s, a_t)\}, \{(a_s, a'_s), (a_t, a'_t)\}, \hat{Z}]$ captures the optimal path from a_s to a_t, if $\hat{Z}$ correctly encodes the adjacent agents to a_s and a_t in an optimal sequence. By iterating over all valid candidates for the neighbors $u \in N_{a_s} \setminus \{a'_s\}$ and $v \in N_{a_t} \setminus \{a'_t\}$ we determine the optimal solution.

To compute the delivery cost of a sequence of agents, we use the predecessor/successor assignment $\hat{A}$. For an agent a of degree 1 we write $\hat{A}(a)$ to denote the predecessor (if a is the start of a path) or successor (if a is the end), and its other adjacent agent by $\hat{Z}(a)$ and the partner of a in the matching M by $M(a)$. We will refer to $\hat{A}(a)$ as the predecessor/successor partner (or *psp* for short) and to $\hat{Z}(a)$ as the next/previous interior partner (or *npip*).

For a feasible forest $S = \{S_1, \ldots, S_b\} \subseteq E_t$, each component $S_i = (a_1, \ldots, a_\ell)$ corresponds to a path of agents starting and ending at agents in B_t^1. Using $\hat{A}$, we extend each such path to the sequence $\hat{S}_i = (\hat{a}_0, a_1, \ldots, a_\ell, \hat{a}_{\ell+1})$, where $\hat{a}_0 = \hat{A}(a_1)$ and $\hat{a}_{\ell+1} = \hat{A}(a_\ell)$. These agents define the handover points at the start and end of the path that involves agents a_1 and a_ℓ, which is necessary to compute the full cost contribution of each agent.

The cost of the extended sequence $\hat{S}_i$ is given by $c(\hat{S}_i) = \sum_{j=1}^{l_i} d_{\hat{a}_j}(D(\hat{a}_{j-1}, \hat{a}_j), D(\hat{a}_j, \hat{a}_{j+1}))$. Setting the predecessor of a_s as a'_s and the successor of a_t as a'_t ensures sequences starting at a_s or ending at a_t are correctly computed as well. In this case we will incur an additional cost of 0, since these agents must immediately handover the package at their respective vertex.

The total cost of the solution S is the sum over all component sequences, i.e., $c(S) := \sum_{i=1}^{b} c(\hat{S}_i)$. In the following we write $\hat{A}(S_i)$ for the extended sequence $\hat{S}_i$. Finally, for any solution $S \subseteq E(G_I)$ and node $t \in V(\mathbb{T})$ we define $S[t] := S \cap E_t$ and say that S *induces* the instance $(t, B_t^0, B_t^1, B_t^2, M, \hat{A}, \hat{Z})$ if

- each $a \in B_t^i$ has degree i in $S[t]$,
- each pair $(u, v) \in M$ is connected by a path in $S[t]$ with $\hat{A}(u), \hat{A}(v)$ as their psp agents, and
- for each npip agent a on a path, the adjacent agent corresponds to $\hat{Z}(a)$.

For an example instance and corresponding agent degrees, see Figure 4. In the following arguments we will for simplicity assume that any DP entry contains the actual forest. If each DP entry only contains its cost one can, by simple recursive pointer-logic, compute the optimal solution in the end recursively. Therefore, we will argue correctness of each entry by considering the actual forest, but derive the computational runtime as if only the cost is saved in every DP entry.

The following lemma shows that indeed it is enough to prove correctness for every subinstance in the DP to derive the optimal solution. The proof is provided in the full version of the paper.

▶ **Lemma 12.** *For every node $t \in V(\mathbb{T})$ and optimal tour $\mathrm{OPT} \subseteq E(G_I)$ that starts with agent a_s and ends with a_t, the partial solution $\mathrm{OPT}[t]$ is optimal for the induced instance.*

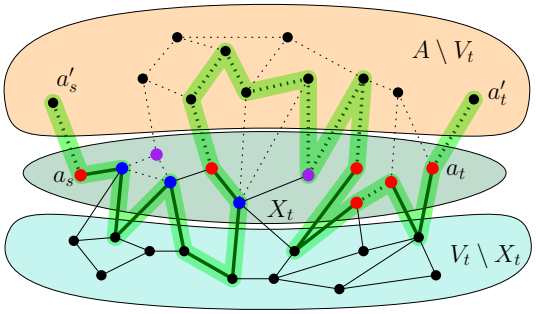


Figure 4 Example instance G_I with corresponding sets of intersections between agents. Agents a_s, a_t are always contained in X_t and a'_s, a'_t are always contained in $A \setminus V_t$. Possible solution ordering of agents shown in green. Red agents have degree 1 in V_t, blue agents have degree 2 and violet agents degree 0. Dashed lines correspond to edges not introduced so far.

We now describe how to handle the different node types in the dynamic program. We will assume that the instance to solve corresponds to a valid specification, otherwise we assign it cost ∞. The proof of correctness for the different node types can be found in the full paper.

Leaf node: At a leaf node t, only a_s, a_t are present and no edges exist. The only valid forest is the empty one with cost 0, so we set $D[t, B_t^0, B_t^1, B_t^2, M, \hat{A}, \hat{Z}] = \{\emptyset\}$.

Introduce node: Let t be an introduce node with child t' and $X_t = X_{t'} \cup \{v\}$. Since v has no incident edges at this point, it must have degree 0 in any valid solution, i.e., $v \in B_t^0$. Therefore v does not influence any solution derived for t' and we set $D[t, B_t^0, B_t^1, B_t^2, M, \hat{A}, \hat{Z}] = D[t', B_t^0 \setminus \{v\}, B_t^1, B_t^2, M, \hat{A}, \hat{Z}]$.

▶ **Lemma 13.** *Let node t introduce an agent v, and let t' be its child. Assume that for all valid combinations the sets $D[t', B_t^{0\prime}, B_t^{1\prime}, B_t^{2\prime}, M', \hat{A}', \hat{Z}']$ have been computed, then we correctly compute $D[t, B_t^0, B_t^1, B_t^2, M, \hat{A}, \hat{Z}]$ in time $O(1)$.*

Forget node: Let t be a forget node with child t' and $X_t = X_{t'} \setminus \{v\}$. Since $v \notin X_t$ and $v \neq a'_s, a'_t$, it must have degree either 0 or 2 in any feasible solution, depending on whether v was used in the respective optimal solution. Therefore, we check both corresponding entries at the child node t' and choose the one with minimum cost.

▶ **Lemma 14.** *Let node $t \in V(\mathbb{T})$ remove some vertex v, and let t' be its child. Assume that for all valid combinations, the sets $D[t', B_t^{0\prime}, B_t^{1\prime}, B_t^{2\prime}, M', \hat{A}', \hat{Z}']$ have been computed, then we correctly compute $D[t, B_t^0, B_t^1, B_t^2, M, \hat{A}, \hat{Z}]$ in time $O(1)$.*

Introduce edge node: Let t be an introduce-edge node with child t' and let $e = \{u, v\}$ be the newly introduced edge. If e is not used in the optimal solution, we reuse the solution from t'. Otherwise, we consider the instance where the degrees of u and v are decreased by 1 and consider all valid choices for their next potential npip agent via $\hat{Z}$. Additionally, we update M accordingly to reflect all possible new pairings that are possible for solutions containing e.

▶ **Lemma 15.** *Let node $t \in V(\mathbb{T})$ introduce an edge $e = \{u, v\}$, and let t' be its child. Assume that for all valid combinations, the sets $D[t', B_t^{0\prime}, B_t^{1\prime}, B_t^{2\prime}, M', \hat{A}', \hat{Z}']$ have been computed, then we correctly compute $D[t, B_t^0, B_t^1, B_t^2, M, \hat{A}, \hat{Z}]$ in time $O((w+1) \cdot \Delta^2)$.*

Join node: Let t be a join node with children t_1 and t_2. To compute an optimal solution, we combine compatible solutions from both children while ensuring degrees, matching, psp agents and npip agents correctly align, and take the solution with minimal cost.

▶ **Lemma 16.** *If node $t \in V(\mathbb{T})$ is a join node with two children $t_1, t_2 \in V(\mathbb{T})$ and we correctly computed all solutions for all child instances for nodes t_1 and t_2, then we correctly compute $D[t, B_t^0, B_t^1, B_t^2, M, \hat{A}, \hat{Z}]$ in time $O((w+2)^{O(\Delta+w)})$.*

Using these lemmata, we establish the final theorem, with the complete proof included in the full version of this work.

▶ **Theorem 3.** *DDT with selectable starting positions can be solved in time $f(\Delta, w) \cdot \text{poly}(n, k)$ on general graphs, where w denotes the treewidth and Δ denotes the maximum degree of the intersection graph and f is a function in w and Δ.*

4.3 A Polynomial-Time Algorithm for Intersection Trees

We now briefly describe an exact algorithm for computing optimal schedules when the intersection graph of the underlying graph forms a tree. While this may seem similar to the setting previously considered, it does not follow directly, as the treewidth may be constant, but the vertex degrees can be unbounded.

We first show how to compute an optimal solution for any instance, given a fixed order of agents $(a_1, \ldots, a_b)$ in the optimal schedule.

For this, we build a layered directed graph G' whose *layers* represent successive handovers:

- Layer 0: single vertex v_s representing the source s.
- Layer i $(1 \leq i \leq b)$: one vertex for every intersection point of the movement areas of a_i and a_{i+1} (for $i = b$ the region of a_{b+1} is the sink region containing t).
- Layer $b+1$: single vertex v_t for the destination t.

Edges connect every vertex in layer $i-1$ to every vertex in layer i; the weight equals the travel time of agent a_i between these two points. Thus each s–t path in G' selects exactly one handover point per layer and has length equal to the total delivery time for that sequence of drones. Conversely, any schedule that uses these agents in this particular order defines such a path. Hence, the shortest path in G' yields the optimal schedule for the given order, which can easily be computed.

▶ **Lemma 17.** *Given any ordered sequence of agents $(a_1, \ldots, a_b)$, we can compute the optimal route to deliver the package from s to t using the agents in this order in $O(b^2 \cdot n^2)$.*

Proof. We construct a directed weighted delivery graph $G' = (V', E')$, $w : E' \to \mathbb{R}_{\geq 0}$. The vertex set V' consists of $b+1$ layers. In layer 0 we have a vertex v_{a_1}, in layer $b+1$ we have a vertex v_t and in layer $i \in [b]$ we have all intersection points between agent a_i and a_{i+1}, i.e., $V_{a_i} \cap V_{a_{i+1}}$. In the following, we refer to these vertices as $V_i = \{v_1^i, \ldots, v_{m_i}^i\}$. We connect v_{a_1} with every vertex v_j^1 for $j \in [m_1]$ and set its weight as $d_{a_1}(s, v_j^1)$. This connection represents the case that agent a_1 transports the package from s to agent a_2 via some intersection point v_j^1. Similarly, we connect a vertex v_j^i , $j \in [m_i]$ in layer i with a vertex $v_{j'}^{i+1}$, $j' \in [m_{i+1}]$, if there exists a path in G_{a_i} that connects v_j^i with $v_{j'}^{i+1}$. For each such edge we again set its weight to the time it takes to get from the first intersection point to the next using agent a_i, i.e., we set the weight of the edge as $d_{a_i}(v_j^i, v_{j'}^{i+1})$. At last, we set the connections from the second last layer b to v_t in the same manner, i.e., we set its weight to $d_{a_b}(v_j^b, t)$, where $j \in [m_b]$.

We want to argue, that the shortest path in this graph has to correspond to an optimal delivery tour of the agents in the specified order. Let $P_{\text{OPT}} \subseteq E$ be the optimal tour and $(p_1^{\text{OPT}} = s, p_2^{\text{OPT}}, \ldots, p_b^{\text{OPT}}, p_{b+1}^{\text{OPT}} = t)$ be the set of vertices at which an exchange of the package happens, including s and t. We can see that for each section $p_i^{\text{OPT}}, p_{i+1}^{\text{OPT}}$, the delivery

happens using agent a_i to agent a_{i+1}. Since the delivery points are specified by $p_i^{\text{OPT}}, p_{i+1}^{\text{OPT}}$, we can upper bound the optimal travel time of a_i for this section by $d_{a_i}(p_i^{\text{OPT}}, p_{i+1}^{\text{OPT}})$. At last, for the final section $p_b^{\text{OPT}}, p_{b+1}^{\text{OPT}} = t$, the delivery time of agent a_b is upper bounded by $d_{a_b}(p_b^{\text{OPT}}, p_{b+1}^{\text{OPT}} = t)$.

We can see that the optimal path $(p_1^{\text{OPT}} = s, p_2^{\text{OPT}}, \ldots, p_b^{\text{OPT}}, p_{b+1}^{\text{OPT}} = t)$ is always a feasible shortest $s - t$ path solution in graph G', and since its cost is at most the cost of the optimal solution we know that the cost of the shortest path tour will be at most the shortest delivery time in the original instance when using the agents in the specified order. On the other hand, every feasible path in our graph corresponds to a unique tour using the same set of agents in the same order and exchanging the package at vertices $p_1 = s, p_2, \ldots, p_b, p_{b+1} = t$. If one such tour would have delivery time strictly less than OPT, this would contradict its optimality assumption, since the vertices on the shortest path also define a feasible drone delivery routing that delivers the package from s to t using the same set of agents in the given order. Therefore it follows, that the shortest path gives us the optimal delivery tour.

Our graph G' consists of at most $(n-1) \cdot b + 2$ vertices and at most $2 \cdot n + (b-2) \cdot n^2$ edges. Using Dijkstra's algorithm with lists, which has runtime $O(n^2)$, we get a final runtime of $O(b^2 \cdot n^2)$. ◀

One could use this property to calculate the optimal schedule for general graphs in time $O(k \cdot k! \cdot k^2 \cdot n^2)$. It also allows us to solve DDT-SP efficiently, if the underlying simple graph of the intersection graph is a tree, in time $O(k^2 \cdot n^2)$. We conclude by proving Theorem 4.

▶ **Theorem 4.** *DDT on a general graph with selectable starting positions can be solved in time $O(k^2 \cdot n^2)$ if the underlying graph of the intersection graph is a tree.*

Proof. Any delivery schedule corresponds to a path in the intersection graph. If this graph is a tree, there always exists a unique simple path between any two agents in G_I. Given the first and last agents used in an optimal schedule, the order of all participating drones can be determined by this path, as each agent appears at most once. Using this order and Lemma 17, we can efficiently compute the delivery schedule. To identify the first and last agents, we consider all valid combinations of agents that can pick up the package at s and drop it off at t. Since each vertex in the graph belongs to at most two agents – otherwise, a clique of size at least three would be present in G_I, contradicting the tree property – there are at most four such combinations. Simply enumerating all four candidate pairs and evaluating each leads to an overall runtime of $O(k^2 \cdot n^2)$. ◀

5 Concluding Remarks and Future Work

We first proved that the Drone Delivery Problem with selectable starting positions (DDT-SP) is $a(n)$–APX-hard even on path graphs, thereby resolving an open question posed by Erlebach et al. and Bartlmae et al. [3,11]. On the algorithmic side, we showed that DDT-SP on a path is fixed-parameter tractable when parameterized solely by the treewidth w of the agents' intersection graph. Employing more sophisticated techniques and parameterizing by both w and the maximum degree Δ of the intersection graph, we further obtained an FPT algorithm for general graphs. When the intersection graph is a tree, a layered-graph formulation yields a polynomial-time exact algorithm.

We believe that the introduction of the intersection graph and its properties is therefore of great value when analyzing the complexity of DDT.

The question of whether DDT on a path is strongly NP-hard remains open for both DDT-FP and DDT-SP. For DDT-SP on a path the complexity for any fixed number of speeds greater than two also remains unresolved. Future work could tighten our parameter

dependencies, or establish stronger lower bounds such as W-hardness. Another promising direction is to analyze DDT under smoothed and semi-random input models, where small random perturbations of worst-case instances provide a more realistic measure of expected computational complexity and can guide the development of practical algorithms.

References

1 Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of Finding Embeddings in a k-Tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987. doi:10.1137/0608024.

2 Dane Bamburry. Drones: Designed for Product Delivery. *Design Management Review*, 26(1):40–48, 2015. doi:10.1111/drev.10313.

3 Simon Bartlmae, Andreas Hene, and Kelin Luo. On the Hardness of the Drone Delivery Problem. In Irene Finocchi and Loukas Georgiadis, editors, *Algorithms and Complexity - 14th International Conference, CIAC 2025, Rome, Italy, June 10-12, 2025, Proceedings, Part II*, volume 15680 of *Lecture Notes in Computer Science*, pages 200–215. Springer, 2025. doi:10.1007/978-3-031-92935-9_13.

4 Andreas Bärtschi, Daniel Graf, and Matús Mihalák. Collective Fast Delivery by Energy-Efficient Agents. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018*, volume 117 of *LIPIcs*, pages 56:1–56:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.MFCS.2018.56.

5 Lotte Blank, Kien C. Huynh, Kelin Luo, and Anurag Murty Naredla. Algorithms for the Collaborative Delivery Problem with Monitored Constraints. In Shin-ichi Nakano and Mingyu Xiao, editors, *WALCOM: Algorithms and Computation - 19th International Conference and Workshops on Algorithms and Computation, WALCOM 2025, Chengdu, China, February 28 - March 2, 2025, Proceedings*, volume 15411 of *Lecture Notes in Computer Science*, pages 62–78. Springer, 2025. doi:10.1007/978-981-96-2845-2_5.

6 Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 226–234. ACM, 1993. doi:10.1145/167088.167161.

7 Heleen Buldeo Rai, Sara Verlinde, and Cathy Macharis. Unlocking the failed delivery problem? Opportunities and challenges for smart locks from a consumer perspective. *Research in Transportation Economics*, 87:100753, 2021. E-groceries, digitalization and sustainability. doi:10.1016/j.retrec.2019.100753.

8 Iago A. Carvalho, Thomas Erlebach, and Kleitos Papadopoulos. On the fast delivery problem with one or two packages. *Journal of Computer and System Sciences*, 115:246–263, 2021. doi:10.1016/J.JCSS.2020.09.002.

9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

10 Mo ElSayed, Ahmed Foda, and Moataz Mohamed. The impact of civil airspace policies on the viability of adopting autonomous unmanned aerial vehicles in last-mile applications. *Transport Policy*, 145:37–54, 2024. doi:10.1016/j.tranpol.2023.10.002.

11 Thomas Erlebach, Kelin Luo, and Frits C. R. Spieksma. Package Delivery Using Drones with Restricted Movement Areas. In Sang Won Bae and Heejin Park, editors, *33rd International Symposium on Algorithms and Computation, ISAAC 2022, December 19-21, 2022, Seoul, Korea*, volume 248 of *LIPIcs*, pages 49:1–49:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ISAAC.2022.49.

12 M. R. Garey and David S. Johnson. Complexity Results for Multiprocessor Scheduling under Resource Constraints. *SIAM J. Comput.*, 4(4):397–411, 1975. doi:10.1137/0204035.

13 Anne Goodchild and Jordan Toy. Delivery by drone: An evaluation of unmanned aerial vehicle technology in reducing CO2 emissions in the delivery service industry. *Transportation Research Part D: Transport and Environment*, 61:58–67, 2018. Innovative Approaches to Improve the Environmental Performance of Supply Chains and Freight Transportation Systems. doi:10.1016/j.trd.2017.02.017.

14 Heather Hulett, Todd G. Will, and Gerhard J. Woeginger. Multigraph realizations of degree sequences: Maximization is easy, minimization is hard. *Oper. Res. Lett.*, 36(5):594–596, 2008. doi:10.1016/J.ORL.2008.05.004.

15 Cornelis Lekkeikerker and Johan Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51(1):45–64, 1962.

16 Kelin Luo, Chenran Yang, Zonghan Yang, and Yuhao Zhang. The subinterval cover problem. In Vincent Chau, Christoph Dürr, Minming Li, and Pinyan Lu, editors, *Frontiers of Algorithmics - 19th International Joint Conference, IJTCS-FAW 2025, Paris, France, June 30 - July 2, 2025, Proceedings*, volume 15828 of *Lecture Notes in Computer Science*, pages 152–165. Springer, 2025. doi:10.1007/978-981-96-8312-3_12.

17 Almodather Mohamed and Moataz Mohamed. Unmanned Aerial Vehicles in Last-Mile Parcel Delivery: A State-of-the-Art Review. *Drones*, 9(6), 2025. doi:10.3390/drones9060413.

18 Neil Robertson and Paul D. Seymour. Graph Minors. II. Algorithmic Aspects of Tree-Width. *J. Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

19 Donald J. Rose, Robert Endre Tarjan, and George S. Lueker. Algorithmic Aspects of Vertex Elimination on Graphs. *SIAM J. Comput.*, 5(2):266–283, 1976. doi:10.1137/0205021.

20 Sartaj Sahni. Algorithms for Scheduling Independent Tasks. *J. ACM*, 23(1):116–127, 1976. doi:10.1145/321921.321934.

21 Frank Schroth. IBM granted patent for package transfer between drones. https://dronelife.com/2017/04/30/ibm-granted-patent-package-transfer-drones/, 2017. [Accessed 23-06-2025].