Collaborative Truck–Multi-Drone Delivery Systems: Scheduling and En-Route Operational Optimization

by
. Jerry Mathew Oommen
10th December 2025

A thesis submitted to the faculty of the Graduate School of the University at Buffalo, The State University of New York in partial fulfillment of the requirements for the degree of
Master of Science
Department of Computer Science and Engineering

Acknowledgments

I would like to thank my advisor, Prof. Kelin Luo, for her guidance and support throughout my thesis. I also thank the faculty and staff of the Department of Computer Science and Engineering at University at Buffalo for their assistance, and my family and friends for their encouragement and support during my graduate studies.

Abstract

Truck–drone delivery systems are increasingly important for last-mile logistics, combining the flexibility of drones with the capacity of trucks. The studied system involves a truck and one or more drones operating in a two-dimensional plane to serve a given set of delivery points: the truck moves along a straight path (the x-axis) at a constant velocity $v_T = 1$, serving as a mobile launch, recovery, and charging station, while drones move freely at velocity v, subject to range R and capacity constraints, to serve off-route customers. Depending on the operational setting, the optimization objective is either to maximize the number of completed deliveries or to minimize the total completion time.

Building on this foundation, this study examines truck–drone delivery systems across three scenarios: (1) multi-drone delivery with randomly located points, (2) multi-drone delivery on structured "proper" instances with geometric constraints, and (3) single-drone delivery where the truck is allowed to stop and wait to support operations. In addition to theoretical analysis, we implement and evaluate several algorithms, including Sequential Greedy, Parallel Greedy, Sequential 1DP, Parallel DP, and Sequential 2DP/2DP* for multi-drone scenarios, as well as heuristic, dynamic programming, and Dijkstra-based exact methods for the truck-stop scenario.

For randomly located points, Sequential Greedy consistently outperforms Parallel Greedy, achieving more deliveries, faster execution, and requiring fewer drones, particularly for smaller fleets. On proper instances, higher-dimensional DP methods (Sequential 2DP*) achieve the largest number of deliveries for small fleets, while Parallel DP scales more efficiently for larger fleets; Sequential 1DP offers a fast alternative with moderate coverage. In the single-drone, truck-stop scenario, a fixed-order bottom-up dynamic programming method minimizes total delivery time for a given delivery order on a discretized state space, while a Dijkstra-based exact search validates solution quality for small instances. The proposed heuristic achieves near-optimal completion times at substantially lower computational cost.

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Background and Motivation

In the last ten years, the use of drones has transformed from test trials to practical tools in delivery operations. This rise in interest is fueled by an increasing need for faster, cheaper and more sustainable solutions for completing last-mile deliveries. There has been a great deal of research into exploring how drones and auxiliary equipment supplement or replace conventional delivery trucks [7, 10].

There are many operational advantages to using drones. They can bypass traffic and save substantial delivery time in congested urban environments because they fly directly between depot and delivery points [7]. This makes them useful for time-critical deliveries such as medicines. Also since they are electrically powered, they consume less energy compared to vehicles powered by internal combustion engines, thus controlling air and gas pollution [9]. Drones also reduce labor costs associated with last-mile deliveries and analyses have shown that they can save up to 96.5% compared with traditional methods [7]. In spite of these benefits, there are still many challenges associated with prevalent use of drone delivery. One of the main hurdles is limited battery endurance and payload capacity, because of which most commercial drones can carry only small packages over short distances [8]. Another major challenge is related to weather which includes strong winds, heavy rain, or extreme temperatures. All of these can cause the battery to drain fast, reduce flight stability and even shut off operations completely that can affect reliability of the service [2].

Flying Sidekick Traveling Salesman Problem (FSTSP) is an efficient truck-drone collaborative model suggested by Murray et al [5]. In this scenario, truck travels in its route, by also functioning as a mobile platform that recharges and reloads the drones. This parallel operation reduces the time required to deliver all packages. Experimental results show such a truck-drone system can reduce delivery time by about 30% to 38% compared to truck-only operations [7]. As an extension to this problem, the Multiple Flying Sidekicks Traveling Salesman Problem (M-FSTSP) is introduced where multiple drones are managed by a single truck, thus increasing its effectiveness [6]. Another recent extension consider the multi-visit drone capability, in which a drone can serve multiple customers on a single sortie before returning to the truck [3].

The specific problem addressed in this study is the En Route Truck-Drone Delivery Problem [4]. This framework models a scenario where the truck travels continuously along a straight line while in parallel dispatches and retrieves a drone that delivers packages to off-route clients. The core objective is optimizing the schedule synchronization between the moving truck and the limited-range drone to maximize the total number of successful deliveries [4]. While the foundational work established strong theoretical results for a single-drone system, practical implementation requires a fleet of drones. Our research extends the entire theoretical and algorithmic framework of the En Route Truck-Drone Delivery Problem to a multi-agent system involving k drones. We derive theoretical performance guarantees for these multi-drone algorithms and validate them through empirical analysis across three distinct logistic scenarios.

1.2 Literature Review

Seidakhmetov and Valilai [8] discuss limitations of drone delivery systems such as battery endurance, payload, regulation, and operational safety constraints. Their work is not mainly an optimization model, but it helps explain why truck–drone cooperation is needed to overcome limited drone range. This is different from routing papers because the focus is on restrictions

and feasibility issues more than algorithm design. These constraints also motivate our scheduling assumptions, especially range and service reliability.

Benarbia and Kyamakya [2] provide a literature review of drone-based package delivery logistics and discuss implementation feasibility. They summarize recurring issues like safety, regulation, reliability, and integration with existing logistics systems. This differs from optimization-focused surveys because it emphasizes practical deployment barriers and system-level challenges. Their discussion supports our motivation that purely theoretical schedules must be interpreted with real operational constraints in mind.

Saeedi et al. [7] present a systematic review on drones in last-mile delivery with emphasis on efficiency and sustainability outcomes. They synthesize evidence on time/cost benefits and also highlight challenges such as accessibility, service reliability, and operational limitations. This differs from Benarbia and Kyamakya [2] because it is more focused on performance and sustainability metrics across studies. Their review motivates why hybrid truck–drone systems can be valuable in practice.

Stolaroff et al. [9] study the energy use and greenhouse gas impacts of drones for commercial package delivery. They show that environmental benefit depends on factors like delivery distance, payload, electricity source, and how drones replace ground vehicle miles. This is different from routing literature because it evaluates environmental outcomes rather than optimizing routes. The paper supports the broader motivation that well-designed drone logistics can have sustainability benefits.

Zhang [10] studies economic and environmental impacts of drone delivery in a thesis-level treatment. The work is broader than one model and discusses tradeoffs between cost, infrastructure, and environmental externalities. This differs from journal routing papers because it provides a long-form analysis across multiple aspects of drone delivery rather than a single algorithmic contribution. It provides background context for why hybrid delivery systems are studied in both industry and academia.

Murray and Chu [5] introduced the Flying Sidekick Traveling Salesman Problem (FSTSP), where a truck follows a route and a drone performs sorties in parallel to reduce total completion time. Their model captures the core coupling between truck movement and drone missions, including the need to synchronize launch and recovery with the truck tour. This differs from later en-route models because rendezvous is typically tied to points along a discrete truck route rather than a continuously moving reference trajectory. FSTSP is important because it defines a standard baseline that many later truck–drone models extend.

Murray and Raj [6] extend the FSTSP to the multiple-drone setting (M-FSTSP), where one truck coordinates more than one drone. The main new difficulty is not only routing but also multi-drone scheduling conflicts and resource sharing (launch/recovery/coordination). This differs from single-drone models because parallel sorties create additional coupling constraints and opportunities for higher throughput. The multi-drone viewpoint is directly relevant to our thesis since our main goal is to extend a single-drone framework to k drones.

Ha et al. [3] study a variant where a drone can serve multiple customers in one sortie (multi-visit), instead of the common one-customer-per-sortie assumption. This changes feasibility because the drone route becomes a small tour rather than a single detour, and it changes how synchronization is handled with the truck. This differs from FSTSP-style models because one sortie is no longer a single customer insertion; it becomes a mini-routing problem inside the global route. It motivates realistic extensions where drone missions are richer than single deliveries, even though our thesis keeps a simpler sortie model.

Agatz et al. [18] summarize the Traveling Salesman Problem with Drone (TSP-D) and related coordinated routing formulations. They explain the typical modeling choices, like

launch/recovery rules, endurance constraints, and objective functions, and they review exact and heuristic solution strategies. This differs from individual algorithm papers because it gives a structured overview of how TSP-D variants relate to each other. This paper helps position our work as part of the broader family of synchronized truck–drone routing and scheduling problems.

Carlsson and Song [19] study coordinated truck–drone logistics from an operations research perspective and emphasize modeling tradeoffs in synchronization. They clarify how meeting constraints and timing assumptions influence system performance and optimal structure. This differs from survey papers because it is not mainly collecting literature; it is analyzing a coordinated system and its decision structure. Their work supports the idea that synchronization assumptions are central, which is also the core difficulty in the En Route model.

Marinelli et al. [13] study truck–drone delivery with continuous en-route operations where drone launch and recovery are allowed on arcs of the truck route, not only at nodes. Their model explicitly includes truck waiting time, which is important because rendezvous timing affects total cost and feasibility. This differs from node-based models because it increases operational freedom but also requires more careful synchronization decisions. Their approach motivates our interest in continuous synchronization, even though our thesis uses a geometric straight-line truck path rather than a general road-network route.

Li et al. [11] consider synchronization on arcs in a road-network formulation, where rendezvous occurs at specified points along road segments. The focus is on capturing arc-level synchronization constraints precisely inside a routing formulation. This differs from Marinelli et al. [13] mainly in how the arc rendezvous is modeled and emphasized (formal arc constraints versus heuristic insertion viewpoint). Their work supports the idea that allowing arc rendezvous can improve efficiency compared with strict node-only synchronization.

Krizanc et al. [4] introduce the En Route Truck–Drone Delivery Problem in a geometric setting where the truck moves continuously along a fixed line and the drone serves off-route points under range constraints. Unlike VRP-D/TSP-D models that choose the truck route, their truck trajectory is fixed, so the main decision is the schedule of drone sorties and rendezvous times. They provide provable algorithmic results, including a greedy 2-approximation and an optimal dynamic programming algorithm for proper instances. This is the closest paper to our work, and our thesis extends this framework from one drone to k drones.

Otto et al. [14] survey optimization approaches for civil UAV applications, including logistics, inspection, and other domains. They summarize major constraint types (battery, payload, safety) and discuss common algorithmic approaches (exact and heuristic). This differs from truck–drone-only surveys because it is broader and not limited to delivery routing. The paper gives context for how delivery fits into the wider UAV optimization literature.

Chung et al. [15] review optimization for drone delivery and for combined drone–truck operations. They organize the literature by objectives and constraints and discuss typical solution methodologies and open challenges such as synchronization and scalability. This differs from Otto et al. [14] because it is more specialized on delivery and hybrid operations. It helps justify why synchronization and coordination constraints are repeatedly central in this field.

Macrina et al. [16] review drone-aided routing problems with focus on TSP-D and VRP-D variants. They compare modeling choices and how different coordination rules change the mathematical structure and difficulty. This differs from Chung et al. [15] because it is more routing-formulation centered and less general about delivery systems. It is useful for positioning our model relative to common routing variants even though our model is geometric and schedule-centered.

Dang et al. [17] survey cooperated truck–drone routing and propose a taxonomy based on

coordination mode, constraints, and objective functions. They explain how different launch/recovery policies and multi-drone assumptions lead to different computational challenges and solution patterns. This differs from Macrina et al. [16] by being more focused specifically on truck–drone cooperation rather than all drone-aided routing. Their taxonomy helps frame our work as a multi-drone, continuous-synchronization style problem.

Boysen et al. [20] study drone delivery from trucks when the truck route is given and the main task is to schedule feasible drone missions. They show that conflicts and endurance constraints create non-trivial scheduling decisions even without truck routing. This differs from full VRP-D models because it isolates the scheduling layer and removes the complexity of truck route selection. This perspective is relevant to our work because the En Route model also fixes the truck path and makes scheduling the main difficulty.

Betti Sorbelli et al. [1] study greedy algorithms for scheduling package delivery with multiple drones on a fixed truck route. Their emphasis is on selecting and timing missions under battery constraints and conflict-free operations, often with reward/coverage-style objectives. This differs from Boysen et al. [20] because it focuses more on greedy algorithm design and performance for the multi-drone setting. It is related to our thesis since we also analyze greedy-style approaches in multi-drone scheduling.

Tamke and Buscher [21] develop an exact branch-and-cut approach for VRP-D variants. Their key contribution is an exact solution methodology that uses strong formulations and cuts to solve instances to optimality when feasible computationally. This differs from heuristic frameworks because it targets provable optimal solutions rather than scalable approximate solutions. Exact baselines like this help show the gap between optimality and heuristics, even though our thesis mainly focuses on approximation and dynamic programming in geometric settings.

Wang and Sheu [22] formulate a VRP with drones and incorporate additional operational constraints affecting launch/recovery and coordination. They show how model realism changes feasibility and also changes the structure of good solutions. This differs from purely theoretical models because it emphasizes practical constraints inside the routing formulation. It is relevant because our work also depends strongly on feasibility constraints (range and synchronization), even though our geometry is simplified.

Zhou et al. [23] consider a two-echelon vehicle routing problem with drones and provide an exact algorithmic approach. The two-level structure represents a richer logistics chain than single-route models and changes how coordination decisions are made. This differs from standard VRP-D because there is an additional distribution layer that affects routing and drone usage. This work shows how truck–drone coordination is studied even in more complex distribution architectures.

Poikonen et al. [24] develop a branch-and-bound approach for the traveling salesman problem with a drone. Their work provides exact optimization for a tour-based coordinated routing model and shows how bounding and search can solve moderate instances optimally. This differs from VRP-D exact work because TSP-D is a single-tour setting rather than multiple-route vehicle routing. It is useful as a reference for exact solution techniques on synchronized routing problems.

Vásquez et al. [25] propose an exact decomposition-based method for TSP-D. The main idea is to split the global decision into structured subproblems (such as truck route and drone sortie selection) to improve solvability. This differs from pure branch-and-bound because decomposition exploits problem structure rather than only search. It motivates using structured dynamic programming and decomposition ideas, which is also important in our proper-instance DP algorithms.

Roberti and Ruthmair [26] study exact methods for TSP-D variants using strengthened formulations and exact optimization strategies. Their work is broader across exact techniques and illustrates how formulation strength affects computational performance. This differs from Vásquez et al. [25] because it is not only one decomposition approach, but more a set of exact modeling and solution improvements. It gives context that exact optimization is possible for some instances, but often becomes expensive as constraints grow.

Thomas et al. [12] study a collaborative truck multi-drone delivery system with $k \geq 2$ drones and integrate scheduling decisions with en-route operations. Their model explicitly handles parallel sorties and coordination constraints that arise when multiple drones share a truck platform. This differs from single-drone coordinated routing because conflicts and resource sharing become central. Their work supports the practical motivation for our multi-drone extension, although our model is geometric and schedule-centered.

Madani et al. [27] study a hybrid truck–drone system where the drone can serve multiple customers per dispatch and launch/recovery locations can be flexible. This increases realism but also increases complexity because one sortie becomes a small routing problem and rendezvous decisions become less structured. This differs from Thomas et al. [12] because the emphasis is on richer sortie patterns rather than mainly multi-drone parallelism. It shows another direction of extension beyond our assumptions, where sorties are not single-customer.

Ren et al. [28] propose a heuristic framework that aims to perform well across multiple VRP-D variants. The focus is on scalability and robustness to different coordination rules and objectives, rather than proving optimality or approximation ratios. This differs from exact methods (e.g., branch-and-cut/branch-and-bound) because it targets large instances and practical solution quality. Such heuristic frameworks are relevant for benchmarking empirical performance when exact methods are not feasible.

Jeong et al. [29] incorporate payload-dependent energy use and no-fly zones into truck–drone routing. They show how realistic constraints like restricted airspace and energy consumption change both feasibility and solution structure. This differs from simplified geometric models because feasibility is shaped by complex geometry and regulation, not only by range and speed. The work motivates why our theoretical results should be interpreted with practical constraints in mind.

Dukkanci et al. [30] study range-limited drone deliveries while optimizing drone speed to reduce energy and cost. Their contribution highlights that operational control variables (like speed) can be optimized together with routing/scheduling decisions. This differs from Jeong et al. [29] because it focuses more on energy-speed tradeoffs rather than airspace restrictions. It motivates future extensions where our model can include speed and energy as decision variables instead of fixed parameters.

Nemhauser et al. [31] provide the classical analysis for greedy approximation when maximizing a monotone submodular function under a cardinality constraint. This is important because maximum coverage is a standard example of such a function, so many greedy bounds follow this framework. This differs from truck–drone papers because it is general approximation theory independent of any routing model. We cite it because our approximation-ratio reasoning connects to coverage-style greedy analysis.

Chvátal [33] analyzes the greedy heuristic for the set cover minimization problem and derives a logarithmic (harmonic) approximation bound. This differs from Nemhauser et al. [31] because it is a minimization problem (cover all elements) instead of a maximization problem (cover as many as possible). It is often used as a baseline theoretical tool when problems have set cover structure. We use it to relate our arguments to classical coverage bounds.

Khuller et al. [34] study budgeted maximum coverage, where each set has a cost and a

budget limits the total cost. They provide approximation guarantees and show how budgets change both the algorithm design and the analysis. This differs from plain maximum coverage because the constraint is not only the number of chosen sets but also a cost budget. This is relevant when deliveries or sorties can be viewed as coverage actions under limited resources.

Feige [32] provides hardness results for approximating set cover and shows that improving substantially beyond logarithmic factors is unlikely under standard complexity assumptions. This is not an algorithmic contribution but a negative result that sets limits on what approximation ratios can be expected. This differs from Chvátal [33] because Chvátal analyzes a greedy algorithm, while Feige establishes near-tight lower bounds on approximability. We cite it to justify why certain coverage-style approximation guarantees are close to the best possible in general.

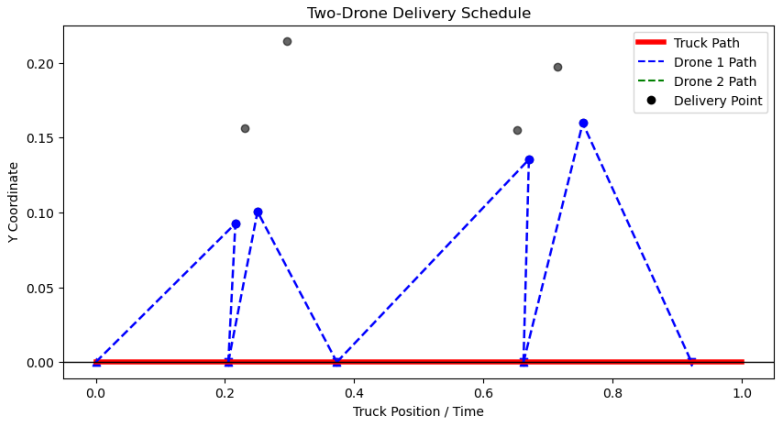


Figure 1: Conceptual diagram of a single-drone mission schedule. The drone launches from the truck (x-axis), delivers a package, and returns to the moving truck.

1.3 Our Problems

We adopt the En Route geometric model and single-drone feasibility definitions from Krizanc et al. [4]. Our original contribution is to generalize scheduling—including greedy heuristics and dynamic programming (DP) approaches to analyze a system involving k drones simultaneously. (illustrated in Figure 2).

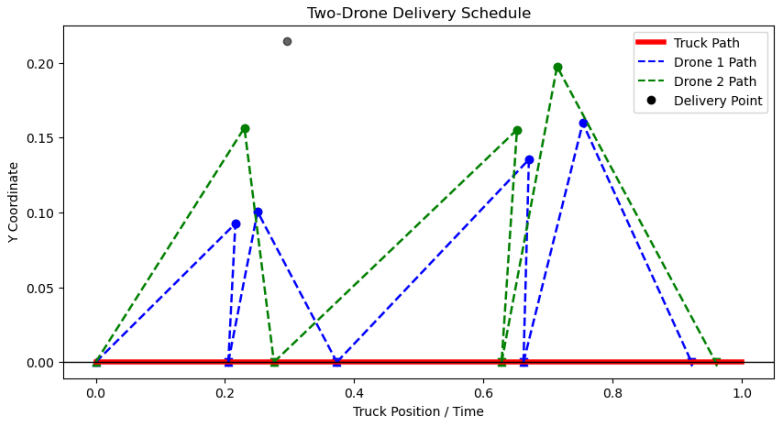


Figure 2: Conceptual diagram of a multi-drone schedule. Two drones are launched from the moving truck to service different delivery points.

The project derived theoretical performance guarantees for these multi-drone algorithms and validated them through extensive empirical analysis across three distinct problem variations:

1.3.1 Truck-Drone Model: k-Drones

This phase addressed the problem of maximizing the total number of deliveries from a set of randomly located points. The general problem formulation was established (Section 2.1). We developed two methods—a sequential greedy and a parallel greedy algorithm—and performed theoretical analysis to determine the minimum approximation factor achievable by the sequential greedy algorithm. Experimental validation confirmed the competitive performance of the algorithms, demonstrating that the sequential greedy algorithm outperformed the parallel greedy algorithm in terms of solution quality.

1.3.2 Truck-Drone Model with Restricted Set of Inputs: k-Drones

This phase focused on the more constrained scenario of proper instances (delivery points conforming to specific geometric rules), building directly on the theoretical framework in [4]. The problem formulation for maximizing deliveries on proper instances was established. We developed four algorithms: Sequential 1DP, Parallel DP, Sequential 2DP and Sequential 2DP*. The theoretical analysis determined the minimum approximation factor achievable by the sequential dynamic programming approach. Empirically, Sequential 2DP* achieved the highest delivery counts for small to moderate fleets, while Parallel DP scaled best and became the top performer as fleet size increased. Sequential 1DP, although not matching the solution quality of the higher-dimensional DP methods, remained computationally feasible and provided a practical, scalable alternative to the prohibitively expensive Sequential 2DP*.

1.3.3 Truck–Drone Model: Truck Can Stop (Single Drone)

In the final phase, the objective was shifted to minimizing the total time required to serve all delivery points using a single drone in a scenario where the truck was allowed to stop for drone operations. The input instances were drawn from the general case, with delivery points placed arbitrarily in the plane. Four solution methods were implemented: a naive approach that schedules deliveries sequentially without optimization, an iterative heuristic algorithm, a fixed-order bottom-up dynamic programming (DP) algorithm, and an order-free exact algorithm based on Dijkstra's shortest-path search. Experimental results showed that the DP algorithm consistently produces the lowest total delivery time among polynomial-time methods, reflecting its exact optimization of discretized launch and return positions for a fixed delivery order. The Dijkstra-based algorithm matches this performance but incurs an exponential increase in execution time, limiting its practicality to small instances. The heuristic method, while slightly suboptimal, achieves near-optimal solutions with significantly lower computational cost. The naive approach performs worse than all optimized methods but provides a baseline for comparison, illustrating the trade-off between solution quality and computational efficiency.

2 Truck-Drone Model: k-Drones

This section initiates the multi-drone setting by extending the single-drone model to a fleet of k drones. The objective in this chapter is coverage: maximize the number of deliveries that can be completed under range and synchronization constraints. We first define the feasibility

notation used throughout, then present the greedy scheduling strategies, and finally evaluate the algorithms on randomized instances.

2.1 Problem Notations

The delivery system operates in the two-dimensional Cartesian plane. The warehouse is located at the origin $[0, 0]$, and the truck starts its journey from there at time 0, fully loaded with all items. Without loss of generality, we assume that the truck travels rightward along the x-axis at a constant speed of 1. This establishes that the time elapsed is equal to the distance the truck has traveled from the origin [4]. The truck travels up to the point called $max\_truck\_distance$.

The drone has a velocity $v > 1$ and can move freely, but its operation is strictly limited by a maximum flying range R due to battery capacity. For simplification, the time required for recharging, picking up, and dropping off items is assumed negligible (zero). We are given a multi-set of delivery points $D = \{d_1, d_2, \ldots, d_n\}$ in the plane where each point is denoted by $d_i = [x_i, y_i]$. Since the truck only delivers to clients on the x-axis, all points in D must be serviced by the drone [4].

2.1.1 Preliminaries

This subsection summarizes feasibility definitions (ellipse parameters and return-time equation) introduced in [4]. We restate them to make the thesis self-contained.

The feasibility of a drone trip is derived from elliptical equations, defining the set of truck locations from which a specific delivery point $d = [x, y]$ can be served within the drone's range R. All parameters and variables in this section are defined according to the reference work [4]. The key geometric parameters are:

$$R = \text{Range of drone}, \qquad v = \text{velocity of drone},$$

$$M = \frac{R}{2}, \qquad m = \frac{R}{2v}\sqrt{v^2 - 1}.$$

For a point $d = [x, y]$, we define an auxiliary variable related to the major axis of the feasibility ellipse:

$$x' = M\left(1 - \frac{y^2}{m^2}\right).$$

Based on these, the earliest/latest feasible times for the drone's launch (s) and return (r) to the truck are defined as follows:

Earliest Start Time ($es(d)$): The earliest time the drone can leave the truck (at position $[es(d), 0]$) to serve delivery point d and still complete the delivery and rendezvous with the moving truck within the maximum range R.

$$es(d) = x - \frac{R}{2v} - x'$$

Earliest Return Time ($er(d)$): The earliest time the drone arrives back at the truck, calculated assuming it launched at the earliest possible time, $es(d)$.

$$er(d) = es(d) + \frac{R}{v}$$

Latest Start Time ($ls(d)$): The latest time the drone can leave the truck (at position $[ls(d), 0]$) to serve delivery point d and still successfully make the rendezvous with the moving truck within the maximum range R.

$$ls(d) = x - \frac{R}{2v} + x'$$

Latest Return Time ($lr(d)$): The latest time the drone arrives back at the truck, calculated assuming it launched at the latest possible time, $ls(d)$.

$$lr(d) = ls(d) + \frac{R}{v}$$

These four parameters define the critical scheduling window within which any feasible drone trip must occur [4].

Drone Return Time Formula The precise time the drone returns to the moving truck, given a starting time s and delivery point d, is calculated by solving the rendezvous equation derived in the reference paper [4]:

$$\mathrm{ret}(s, v, d) := s + \frac{s + av - x + \sqrt{b^2 - s(v^2 - 1)\big(b + s + av - x\big)}}{v^2 - 1},$$

where the auxiliary variables are:

$$a = y^2 + (s - x)^2, \qquad b = sv^2 + av - x.$$

Feasible Delivery Schedule Given an instance $I = (v, R, D)$, a delivery schedule S_I is an ordered list of deliveries, defined as:

$$S_I = ((d_{i_1}, s_1), (d_{i_2}, s_2), \ldots, (d_{i_m}, s_m)), \quad m \leq n$$

Here:

- d_{i_j} is a delivery point served by the drone.
- s_j is the time at which the drone leaves the truck, which is then located at position $[s_j, 0]$.
- m is the total number of deliveries in the schedule (Note: This variable m is distinct from the geometric constant m defined earlier).

The schedule is considered feasible if [4]:

- $s_1 \geq 0$, and
- For each j from 1 to $m - 1$, the drone's return time after delivery d_{i_j}, calculated using $\mathrm{ret}(s_j, v, d_{i_j})$, must be less than or equal to the start time of the next delivery, s_{j+1}.

2.1.2 Greedy Algorithm

First we state the Greedy Algorithm for the truck-drone model with one drone for completeness. This is the single-drone greedy algorithm (A_g) from [4], we include it only as a baseline subroutine for our multi-drone extensions. The Greedy Approximation Algorithm (A_g), foundational to the single-drone model by Krizanc et al. [4], is an $O(n^2)$ heuristic that provides

a practical schedule with a guaranteed 2-approximation to the optimal number of deliveries. Our objective is to adapt and extend this methodology for a system involving k drones. The core logic operates in two phases: first, Pre-filtering, where for every potential delivery, the earliest (es) and latest (ls) launch times are calculated based on the drone's range limit (R) and speed (v). Deliveries whose ls is less than the truck's start time ($s = 0$) are discarded, and the remaining feasible deliveries are placed in set L and sorted by their es. Second, Iterative Scheduling begins by checking the current truck time (s); if s is earlier than the next available delivery's es, s is advanced to that es. From all deliveries currently launchable ($s \geq es$), the algorithm greedily selects the delivery that minimizes the rendezvous (return) time ($r_{\min}$) at the moving truck. The selected delivery is added to the schedule, the truck time s is advanced to $r_{\min}$, and the set L is immediately pruned to remove the chosen delivery and any others whose ls has now been exceeded ($ls < s$). This process iterates, ensuring a non-overlapping drone schedule while locally maximizing the forward progress of the system.

2.2 Algorithms and Analysis

The feasibility windows and return-time function defined above allow us to test whether a candidate delivery can be served by a drone without violating range or rendezvous constraints. Using these primitives, we now design greedy multi-drone scheduling rules whose goal is to increase coverage as quickly as possible while keeping coordination overhead low.

Common Inputs
The algorithms share a set of common inputs:

- **deliveries**: A list of delivery points in the form $[(x_1, y_1), (x_2, y_2), \dots]$.
- **R**: The maximum range the drone can travel on a single trip.
- **v**: The velocity of the drone (v_d).
- **k**: Number of the drones.
- **max_truck_distance**: The maximum endpoint of the route, representing the farthest point the truck must reach.

2.2.1 Algorithm 1 - Sequential Greedy Algorithm

Algorithm Description: Algorithm 1 is a new multi-drone extension proposed in this thesis. The algorithm extends the single-drone greedy strategy to multiple drones by assigning deliveries one drone at a time. Unlike [4] (single-drone), we allocate deliveries across drones by iteratively solving a single-drone subproblem on the remaining points. In each step, the algorithm identifies a feasible set of delivery points that can be served by the current drone using the same greedy selection rule as in the single-drone case. Once the drone's route is determined, those delivery points are removed from consideration, and the process repeats for the next drone with the remaining unassigned points. This continues until all k drones have been assigned or there are no deliveries left. Because each drone operates on the reduced set of points after previous assignments, the algorithm tends to allocate deliveries closer to the x-axis first. The overall running time is $O(kn^2)$, where n is the total number of delivery points and k is the number of drones. The complete pseudocode for the Sequential Greedy Algorithm is presented in Algorithm 1.

Algorithm 1: Sequential Greedy Scheduling Algorithm

Input: Ordered delivery points $deliveries$, Range R, Velocity v, Number of drones num_drones and Maximum Truck Distance x_max

Output: Drone Schedules SI_list, Points that are not possible to be assigned L_not, Points that are remaining to be assigned L

$s \leftarrow 0$ $L \leftarrow \emptyset$ $L\_not \leftarrow \emptyset$ $SI\_list \leftarrow \emptyset$

foreach $d \in deliveries$ **do**

  $(es, ls) \leftarrow \texttt{calculate\_es\_ls}(R, v, d)$

  if (es, ls) *exists* **then**

    if $s \leq ls$ **then**

      $L \leftarrow L \cup \{(es, ls, d)\}$

    else

      $L\_not \leftarrow L\_not \cup \{(es, ls, d)\}$

Sort L by es increasing

for $i \leftarrow 1$ **to** num_drones **do**

  $s \leftarrow 0$

  $SI \leftarrow \emptyset$

  $L\_next \leftarrow \emptyset$

  while $L \neq \emptyset$ ***and*** $s < x\_max$ **do**

    $x \leftarrow$ first element of L

    if $s < x.es$ **then**

      $s \leftarrow x.es$

    $rmin \leftarrow \infty$

    $save \leftarrow$ null

    foreach $x' \in L$ ***such that*** $s \geq x'.es$ **do**

      $r \leftarrow \texttt{ret}(s, v, x'.d)$

      if $r < rmin$ **then**

        $rmin \leftarrow r$

        $save \leftarrow x'$

    if $save \neq$ *null* **then**

      Append $(save.d, s)$ to SI

      $s \leftarrow rmin$

      $L\_next \leftarrow L\_next \cup \{x' \in L \mid x'.ls < s \wedge x'.d \neq save.d\}$

      $L \leftarrow \{x' \in L \mid x'.ls \geq s \wedge x'.d \neq save.d\}$

  $L \leftarrow L\_next$

  Sort L by es increasing

  Append SI to SI_list

return (SI_list, L_not, L)

Approximation Ratio Analysis for Sequential Greedy Algorithm

The approximation bound below is derived for our multi-drone greedy construction. The proof adapts the classical maximum-coverage greedy analysis (e.g., Nemhauser et al. [31]) to the setting where each 'set' corresponds to a feasible single-drone schedule computed by an α-approximation subroutine.

Useful Parameters:

- V_0: The set of all delivery points.
- k: Number of drones.
- S_i: Set of points delivered by the i-th drone in the greedy schedule.
- OPT_i: Set of points delivered by the i-th drone in the optimal schedule.
- $S(k)$: Total number of unique points delivered by k drones in the greedy schedule.
- $OPT(k)$: Total number of unique points delivered by k drones in the optimal schedules.
- V_{i-1}: The set of remaining unserved delivery points at the start of iteration i, defined as the initial set V_0 excluding all points served by the previous $i-1$ greedy drone schedules:

$$V_{i-1} = V_0 \setminus \bigcup_{j=1}^{i-1} S_j.$$

- α: The *approximation factor* $\alpha \in (0, 1]$ is defined as the smallest constant such that

$$S(1) \geq \alpha\, OPT(1)$$

Observation (Disjointness and Total Value). Since each delivery point can be assigned to at most one drone, all individual drone schedules are disjoint:

$$S_i \cap S_j = \emptyset \quad \text{and} \quad OPT_i \cap OPT_j = \emptyset \quad \forall i \neq j.$$

Consequently, the total value of any solution is the sum of its disjoint schedule sizes:

$$S(k) = \sum_{i=1}^{k} |S_i|, \qquad OPT(k) = \sum_{i=1}^{k} |OPT_i|.$$

Lemma 1 (Lower Bound for S_1). Let $OPT_1, \ldots, OPT_k$ be the k disjoint optimal drone schedules, and let $|OPT_{\max}| = \max_i |OPT_i|$ denote the largest single schedule.
By definition, the size of the optimal single schedule on the full set of points satisfies

$$OPT(1) \geq |OPT_{\max}|,$$

because it can always choose the largest schedule among the k disjoint optimal schedules.
Moreover, since the largest schedule is at least as large as the average of all k schedules, we have

$$OPT(1) \geq |OPT_{\max}| \geq \frac{1}{k} \sum_{i=1}^{k} |OPT_i| = \frac{1}{k}\, OPT(k).$$

Consequently, the first greedy drone schedule S_1 satisfies

$$|S_1| \geq \alpha\, OPT(1) \geq \frac{\alpha}{k}\, OPT(k),$$

For the specific Greedy Approximation Algorithm (A_g) where $\alpha = 1/2$ [4], this gives

$$|S_1| \geq \frac{1}{2k} OPT(k).$$

Lemma 2 (Local Approximation Property). At each iteration i, the algorithm selects a drone schedule S_i computed on the remaining set V_{i-1}. Let $OPT_1(V_{i-1})$ denote an optimal single-drone schedule restricted to V_{i-1}. By assumption, the schedule returned at iteration i is an α-approximation to this restricted optimum:

$$|S_i| \geq \alpha \, |OPT_1(V_{i-1})|.$$

For the specific Greedy Approximation Algorithm (A_g) where $\alpha = 1/2$ [4], each choice satisfies

$$|S_i| \geq \frac{1}{2} |OPT_1(V_{i-1})|.$$

Lemma 3 (Case $k = 2$). For two drones, the sequential greedy algorithm satisfies

$$S(2) \geq \frac{7}{16} OPT(2).$$

Proof. Let S_1 be the first greedy schedule. We define the intersections between the first greedy schedule and the two optimal schedules (OPT_1 and OPT_2) as:

$$\begin{aligned} OPT_{1,1} &= S_1 \cap OPT_1, \qquad & OPT_{1,2} &= OPT_1 \setminus S_1, \\ OPT_{2,1} &= S_1 \cap OPT_2, \qquad & OPT_{2,2} &= OPT_2 \setminus S_1. \end{aligned}$$

By the properties of disjointness and total value (Observation), the remaining uncovered optimal points after the first step are $OPT_{1,2} \cup OPT_{2,2}$, with

$$|OPT_{1,2}| + |OPT_{2,2}| \geq OPT(2) - |S_1|.$$

Lower Bound for S_2. Using the local α-approximation property from Lemma 2, where $\alpha = 1/2$ (derived from the 2-approximation guarantee of the single-drone algorithm):

$$|S_2| \geq \frac{1}{2} |OPT_1(V_0 \setminus S_1)|.$$

The best remaining schedule $OPT_1(V_0 \setminus S_1)$ must cover at least as many points as the larger of the two remaining optimal subsets:

$$|OPT_1(V_0 \setminus S_1)| \geq \max(|OPT_{1,2}|, |OPT_{2,2}|).$$

Using $\max(a, b) \geq \frac{1}{2}(a + b)$ gives

$$|S_2| \geq \frac{1}{4}\big(|OPT_{1,2}| + |OPT_{2,2}|\big) \geq \frac{1}{4}\big(OPT(2) - |S_1|\big). \tag{1}$$

Combine the Bounds. Summing $|S_1|$ and Equation (1), yields the recurrence relation:

$$S(2) = |S_1| + |S_2| \geq |S_1| + \frac{1}{4} OPT(2) - \frac{1}{4} |S_1| = \frac{1}{4} OPT(2) + \frac{3}{4} |S_1|. \tag{2}$$

Lower Bound for S_1. From Lemma 1, for general k,

$$|S_1| \geq \frac{\alpha}{k} OPT(k).$$

Setting $\alpha = 1/2$ and $k = 2$ gives the specific lower bound:

$$|S_1| \geq \frac{1}{2 \cdot 2} OPT(2) = \frac{1}{4} OPT(2).$$

Final Substitution. Substituting this lower bound for $|S_1|$ into the recurrence relation (2):

$$\begin{aligned} S(2) &\geq \frac{1}{4} OPT(2) + \frac{3}{4} \left(\frac{1}{4} OPT(2) \right) \\ &= \left(\frac{4}{16} + \frac{3}{16} \right) OPT(2) \\ &= \frac{7}{16} OPT(2). \end{aligned}$$

■

Lemma 4 (Case $k = 3$). For three drones, the sequential greedy algorithm satisfies

$$S(3) \geq \frac{91}{216} OPT(3).$$

Proof. Let the optimal schedules be OPT_1, OPT_2, OPT_3 (pairwise disjoint), and let S_1, S_2, S_3 be the three greedy schedules chosen in order.

Intersections after S_1. Define, for each $i \in \{1, 2, 3\}$,

$$OPT_{i,1} = S_1 \cap OPT_i, \qquad OPT_{i,2} = OPT_i \setminus OPT_{i,1}.$$

By the properties of disjointness and total value (Observation),

$$|OPT_{1,2}| + |OPT_{2,2}| + |OPT_{3,2}| \geq OPT(3) - |S_1|.$$

Bound for S_2. By the local α-approximation property from Lemma 2, where $\alpha = 1/2$, applied to the remaining set $V_0 \setminus S_1$,

$$|S_2| \geq \frac{1}{2} |OPT_1(V_0 \setminus S_1)| \geq \frac{1}{2} \max\{|OPT_{1,2}|, |OPT_{2,2}|, |OPT_{3,2}|\}.$$

Using $\max\{a, b, c\} \geq \frac{1}{3}(a + b + c)$ we get

$$|S_2| \geq \frac{1}{6} \big(|OPT_{1,2}| + |OPT_{2,2}| + |OPT_{3,2}| \big) \geq \frac{1}{6} \big(OPT(3) - |S_1| \big).$$

Hence

$$|S_2| \geq \frac{1}{6} OPT(3) - \frac{1}{6} |S_1|. \tag{3}$$

Combine S_1 and S_2. Summing $|S_1|$ and the bound (3), gives

$$|S_1| + |S_2| \geq |S_1| + \frac{1}{6} OPT(3) - \frac{1}{6} |S_1| = \frac{1}{6} OPT(3) + \frac{5}{6} |S_1|.$$

Lower bound for S_1. From Lemma 1, for $k = 3$ and $\alpha = 1/2$,

$$|S_1| \geq \frac{\alpha}{k} OPT(k) = \frac{1}{6} OPT(3).$$

Substitution for $S_1 + S_2$. Substituting the lower bound for $|S_1|$ into the combined inequality yields the intermediate bound for the first two schedules:

$$|S_1| + |S_2| \geq \frac{1}{6} OPT(3) + \frac{5}{6} \cdot \frac{1}{6} OPT(3) = \left(\frac{6}{36} + \frac{5}{36} \right) OPT(3) = \frac{11}{36} OPT(3).$$

Bound for S_3. Define the remaining optimal sets after $S_1 \cup S_2$:

$$OPT'_{i,1} = (S_1 \cup S_2) \cap OPT_i, \qquad OPT'_{i,2} = OPT_i \setminus OPT'_{i,1}.$$

By disjointness and Total Value (Observation),

$$|OPT'_{1,2}| + |OPT'_{2,2}| + |OPT'_{3,2}| \geq OPT(3) - (|S_1| + |S_2|).$$

Applying the local $\alpha = 1/2$ approximation (Lemma 2) on the remaining set $V_0 \setminus (S_1 \cup S_2)$ and using $\max\{a, b, c\} \geq \frac{1}{3}(a + b + c)$, we obtain

$$|S_3| \geq \frac{1}{6} OPT(3) - \frac{1}{6}\big(|S_1| + |S_2|\big). \tag{4}$$

Combine S_1, S_2 and S_3. Summing $|S_1| + |S_2|$ and the bound (4), and using the Total Value property (Observation), gives the total greedy size $S(3)$:

$$S(3) = |S_1| + |S_2| + |S_3| \geq \frac{1}{6} OPT(3) + \frac{5}{6}\big(|S_1| + |S_2|\big). \tag{5}$$

Final Numeric Substitution. Substituting the intermediate bound $|S_1| + |S_2| \geq \frac{11}{36} OPT(3)$ into Equation (5) gives the final approximation ratio:

$$\begin{aligned} S(3) &\geq \frac{1}{6} OPT(3) + \frac{5}{6} \cdot \frac{11}{36} OPT(3) \\ &= \left(\frac{1}{6} + \frac{55}{216} \right) OPT(3) \\ &= \left(\frac{36}{216} + \frac{55}{216} \right) OPT(3) \\ &= \frac{91}{216} OPT(3). \end{aligned}$$

which proves the claim. ■

Theorem 2.1. *(General case for k drones) Let $S(k)$ denote the total number of deliveries achieved by the sequential greedy algorithm using k drones, and let $OPT(k)$ denote the size of an optimal set of k disjoint schedules. Then we have*

$$S(k) = \sum_{i=1}^{k} |S_i| \geq \left(1 - \left(1 - \frac{\alpha}{k} \right)^k \right) OPT(k).$$

Proof. **Bound for the first schedule.** From the Lower Bound for S_1 in Lemma 1, the first greedy schedule satisfies

$$|S_1| \geq \frac{\alpha}{k} OPT(k).$$

Bounds for subsequent schedules.
Fix an iteration $i \in \{2, \ldots, k\}$. Let

$$V_{i-1} = V_0 \setminus \bigcup_{j=1}^{i-1} S_j$$

be the remaining points after the first $i-1$ greedy selections. By Lemma 2 (Local Optimality) the greedy choice satisfies

$$|S_i| \geq \alpha \, |OPT_1(V_{i-1})|,$$

where $OPT_1(V_{i-1})$ denotes the optimal single schedule on V_{i-1}.

To relate $|OPT_1(V_{i-1})|$ to $OPT(k)$, consider the k disjoint optimal schedules $OPT_1, \ldots, OPT_k$, After the first $i-1$ greedy schedules, the number of points in the remaining portions of the optimal schedules satisfies

$$\sum_{\ell=1}^{k} |OPT_\ell \setminus \bigcup_{j=1}^{i-1} S_j| = \sum_{\ell=1}^{k} |OPT_\ell| - \sum_{\ell=1}^{k} |OPT_\ell \cap \bigcup_{j=1}^{i-1} S_j|.$$

Since the schedules S_j are disjoint, we have

$$\sum_{\ell=1}^{k} |OPT_\ell \cap \bigcup_{j=1}^{i-1} S_j| \leq \sum_{j=1}^{i-1} |S_j|.$$

Combining these gives the key inequality:

$$\sum_{\ell=1}^{k} |OPT_\ell \setminus \bigcup_{j=1}^{i-1} S_j| \geq OPT(k) - \sum_{j=1}^{i-1} |S_j|.$$

Since the value of the largest of these k disjoint subsets must be at least as large as their average, we have:

$$\max_{\ell} |OPT_\ell \setminus \bigcup_{j=1}^{i-1} S_j| \geq \frac{1}{k}\Big(OPT(k) - \sum_{j=1}^{i-1} |S_j|\Big).$$

The best possible single schedule on the remaining points, $OPT_1(V_{i-1})$, cannot do worse than this maximum. Hence

$$|OPT_1(V_{i-1})| \geq \frac{1}{k}\Big(OPT(k) - \sum_{j=1}^{i-1} |S_j|\Big).$$

Applying the α-approximation guarantee to this term yields

$$|S_i| \geq \frac{\alpha}{k}\Big(OPT(k) - \sum_{j=1}^{i-1} |S_j|\Big).$$

In particular:

$$|S_2| \geq \frac{\alpha}{k}\big(OPT(k) - |S_1|\big), \qquad |S_3| \geq \frac{\alpha}{k}\big(OPT(k) - (|S_1| + |S_2|)\big),$$

and so on, up to

$$|S_k| \geq \frac{\alpha}{k}\left(OPT(k) - \sum_{j=1}^{k-1} |S_j|\right).$$

Iterative expansion. Adding the inequalities iteratively, we obtain

$$\begin{aligned}
|S_1| + |S_2| &\geq \frac{\alpha}{k}\Big(OPT(k) + (\frac{k}{\alpha} - 1)|S_1|\Big), \\
|S_1| + |S_2| + |S_3| &\geq \frac{\alpha}{k}\Big(OPT(k) + (\frac{k}{\alpha} - 1)(|S_1| + |S_2|)\Big), \\
&\vdots \\
S(k) = \sum_{i=1}^{k} |S_i| &\geq \frac{\alpha}{k}\left(OPT(k) + (\frac{k}{\alpha} - 1)\sum_{i=1}^{k-1} |S_i|\right),
\end{aligned}$$

where the final step follows from the Observation, which ensures that the greedy schedules S_i are disjoint and additive.

Closed-form expression. Expanding this recursively leads to the geometric series

$$S(k) \geq \frac{\alpha}{k} OPT(k) \left[1 + \left(1 - \frac{\alpha}{k}\right) + \left(1 - \frac{\alpha}{k}\right)^2 + \cdots + \left(1 - \frac{\alpha}{k}\right)^{k-1}\right].$$

Summing the series gives the generalized approximation ratio:

$$S(k) \geq \frac{\alpha}{k} OPT(k) \cdot \frac{1 - \left(1 - \frac{\alpha}{k}\right)^k}{1 - \left(1 - \frac{\alpha}{k}\right)} = OPT(k)\left(1 - \left(1 - \frac{\alpha}{k}\right)^k\right).$$

□

Substituting the specific factor $\alpha = 1/2$ into this generalized closed form yields:

$$S(k) \geq OPT(k)\left(1 - \left(1 - \frac{1}{2k}\right)^k\right).$$

Remark. This proof mirrors the classical analysis of the greedy algorithm for the Maximum Coverage problem, where one of the k optimal sets covers at least a $\frac{1}{k}$ fraction of the remaining elements. Here, each schedule S_i acts as a greedy-chosen set. The Disjointness Constraint (each point served once) ensures that both the greedy schedules ($\{S_1, \ldots, S_k\}$) and optimal schedules ($\{OPT_1, \ldots, OPT_k\}$) form partitions of their served points, making total coverage additive. The α factor arises because each local subproblem (finding S_i) uses an α-approximation algorithm, leading to $|S_i| \geq \frac{\alpha}{k}\big(OPT(k) - \sum_{j=1}^{i-1} |S_j|\big)$, which embeds the α penalty into the classical $\frac{1}{k}$ gain.

Asymptotic Behavior When k tends to infinity, we use the limit $\lim_{n\to\infty}\left(1 + \frac{x}{n}\right)^n = e^x$. Here,

let $n = k$ and $x = -\alpha$.

$$\lim_{k\to\infty}\left(1-\frac{\alpha}{k}\right)^k = \lim_{k\to\infty}\left(\left(1+\frac{-\alpha}{k}\right)^k\right) = e^{-\alpha}$$

So, the approximation factor approaches:

$$\lim_{k\to\infty}\left[1-\left(1-\frac{\alpha}{k}\right)^k\right] = 1-e^{-\alpha}$$

For the specific algorithm studied in the paper [4], where the local approximation factor is $\alpha = 1/2$, the asymptotic bound is:

$$1 - e^{-1/2} \approx 1 - 0.60653 \approx 0.39347.$$

This asymptotic behavior is illustrated by the curve shown in Figure 3.

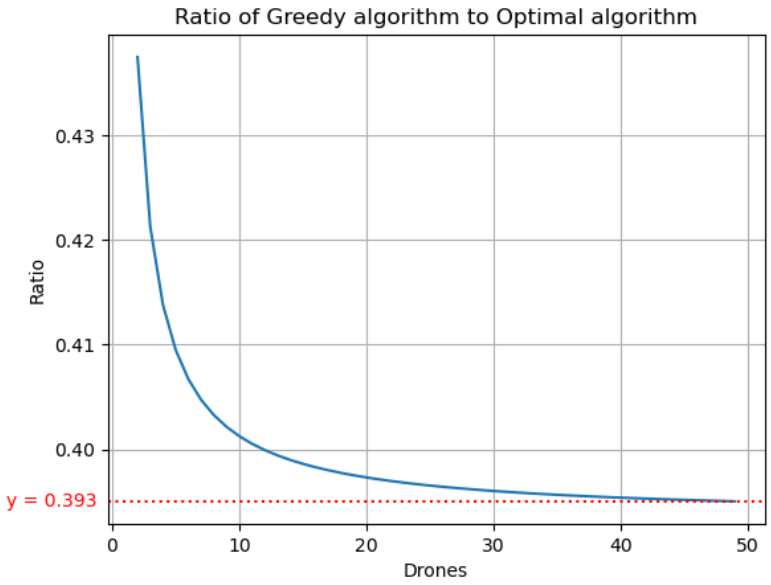


Figure 3: Approximation factor $S(k)/OPT(k)$ as $k \to \infty$ when $\alpha = 1/2$.

2.2.2 Algorithm 2 - Parallel Greedy Algorithm

Algorithm Description: The Parallel Greedy Algorithm schedules deliveries in a round-robin fashion. Unlike the sequential greedy approach, where each drone is fully assigned its route before moving to the next, the parallel greedy algorithm distributes delivery points among all drones in an iterative manner. At each step, every drone is considered once, and a feasible delivery point is selected for it based on the earliest-start and latest-start constraints. This process continues in rounds until no more feasible assignments can be made for any drone. In contrast to the sequential approach—which often allocates nearby deliveries to the first drones and leaves the distant ones for later—the parallel strategy balances the workload across all drones. The time complexity of the algorithm is $O(kn^2)$, where n is the total number of delivery points and k is the number of drones involved in the operation. Algorithm 2 is proposed in this thesis as an alternative multi-drone extension that balances assignments across drones; it is

not part of [4]. This algorithm is heuristic; we do not claim a formal approximation ratio for it in this thesis. The complete pseudocode for the Parallel Greedy Algorithm is presented in Algorithm 2.

Algorithm 2: Parallel Greedy Scheduling Algorithm

Input: Ordered delivery points $deliveries$, Range R, Velocity v, Number of drones num_drones and Maximum Truck Distance x_max

Output: Drone Schedules SI_list, Points that are not possible to be assigned L_not, Points that are remaining to be assigned L

$L \leftarrow \emptyset$ $L\_not \leftarrow \emptyset$ $L\_save \leftarrow \emptyset$ $s[k] \leftarrow 0$ for $k = 1 \dots num\_drones$
foreach $d \in deliveries$ **do**
    $(es, ls) \leftarrow$ `calculate_es_ls`(R, v, d)
    if (es, ls) ***exists*** **then**
        if $0 \leq ls$ **then**
            $L \leftarrow L \cup \{(es, ls, d)\}$
        else
            $L\_not \leftarrow L\_not \cup \{(es, ls, d)\}$
Sort L by es increasing
$SI \leftarrow$ list of num_drones empty lists
$L\_lst \leftarrow$ list of num_drones copies of L
$flag \leftarrow$ True
while $flag =$ ***True*** **do**
    $flag \leftarrow$ False
    for $k \leftarrow 1$ **to** num_drones **do**
        if $L\_lst[k] \neq \emptyset$ ***and*** $s[k] < x\_max$ **then**
            $flag \leftarrow$ True
            $x \leftarrow$ first element of $L\_lst[k]$
            if $s[k] < x.es$ **then**
                $s[k] \leftarrow x.es$
            $rmin \leftarrow \infty$
            $save \leftarrow$ null
            foreach $y \in L\_lst[k]$ ***such that*** $s[k] \geq y.es$ **do**
                $r \leftarrow$ `ret`$(s[k], v, y.d)$
                if $r < rmin$ **then**
                    $rmin \leftarrow r$
                    $save \leftarrow y$
            if $save \neq$ ***null*** **then**
                Append $(save.d, s[k])$ to $SI[k]$
                Append $save$ to L_save
                $s[k] \leftarrow rmin$
                $L\_lst[k] \leftarrow \{x \in L\_lst[k] \mid x.ls \geq s[k]\}$
                for $i \leftarrow 1$ **to** num_drones **do**
                    $L\_lst[i] \leftarrow \{x \in L\_lst[i] \mid x.d \neq save.d\}$
$L\_rem \leftarrow L - L\_save$
return (SI, L_not, L_rem)

2.3 Experiment

To extend the basic greedy algorithm to the multi-drone scenario, two strategies were investigated: Sequential Greedy assignment and Parallel Greedy assignment.

2.3.1 Experimental Setup

To rigorously evaluate the performance of the proposed multi-drone scheduling strategies, a series of computational experiments was conducted. The primary focus was on comparing the efficiency and resource usage of the Sequential Greedy and Parallel Greedy algorithms.

- **Delivery Points and Constraints:** Delivery locations were generated randomly within a two-dimensional plane, represented as (x_i, y_i). To ensure that every point could be feasibly served by a drone, the vertical coordinate (y_i) was constrained by the maximum value given by the equation:

  $$y_{\text{max}} = \frac{R}{2v}\sqrt{v^2 - 1}$$

  This constraint guarantees a drone's round trip from the truck is possible within its maximum range R and velocity v. Multiple sets of randomized delivery points were used across trials.

- **Truck and Drone Parameters:**
  - The truck travels horizontally along the x-axis with a fixed velocity of 1 unit per time unit. The total travel length is defined as $max\_truck\_distance$.
  - The drone moves at a fixed velocity v_d.
  - Each drone has a maximum range R, which constrains its individual mission distance.

- **Hardware and Software Environment:**
  - The algorithms were implemented in Python.
  - Execution was performed on the same hardware platform to ensure a fair comparison: a MacBook Air M2 with an 8-core CPU and 16GB of unified memory.

- **Evaluation Metrics:** Performance was primarily assessed using the following metrics:
  1. *Number of Deliveries*: The count of delivery points successfully served.
  2. *Ratio of Deliveries (S/P)*: The ratio of Sequential to Parallel deliveries, where a value > 1.0 indicates superior performance by the Sequential algorithm.
  3. *Execution Time*: The CPU time required to compute the schedule.
  4. *Resource Efficiency*: The minimum number of drones required to complete a given set of deliveries.

- **Experimental Variations and Performance Analysis:** The following parameters were systematically varied to assess their impact on algorithm performance, particularly the S/P Ratio, as detailed in Table 1:

Table 1: Simulation parameters for experiment 1

Parameter	**Values**
Number of Delivery Points	100, **200**, 300, 400, 500, 600, 700, 800, 900, 1000
Drone Velocity	1.2, 1.4, **1.6**, 1.8, 2.0, 2.2, 2.4, 2.6, 2.8, 3.0
Drone Range	0.2, 0.4, **0.6**, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0
Number of Drones	1, 2, 3, 4, 5, 6, 7, 8, 9, 10

- **Repetition and Averaging:** For each configuration, 100 random instances were generated, and the results were averaged to ensure statistical reliability.
- **Experimental Workflow:** The process for each configuration involved generating the same random set of points, executing both the Sequential and Parallel algorithms, recording the required metrics, and repeating the process to generate reliable performance data.
- **Github link:** Code and results for all the algorithms and experiments presented in this thesis can be accessed at the following link: https://github.com/JerryMathew07/Thesis.

2.3.2 Experimental Results

The greedy algorithms defined above are heuristic by design, so experimental evaluation is necessary to understand the practical trade-offs between solution quality (deliveries completed), computation time, and drone utilization. This section describes the instance generation and metrics, followed by comparative results.

1. Number of Deliveries Comparison
Figure 4 shows the number of deliveries completed as the number of drones increases. The Sequential Greedy Algorithm consistently achieves higher deliveries compared to the Parallel Greedy Algorithm, especially when the number of drones is small. As the number of drones increases, both algorithms approach near-complete coverage, but the sequential method remains slightly more efficient. This reflects the advantage of assigning one drone fully before moving to the next, ensuring maximal feasible deliveries per schedule.

Figure 4: Sequential Greedy algorithm makes more deliveries for a given number of drones.

2. Ratio of deliveries comparison

Figure 5, Figure 6, Figure 7 presents 3D surface plots of the ratio between the number of deliveries made by the Sequential Greedy algorithm and the Parallel Greedy algorithm. The x-axis represents the number of drones (k), while the z-axis shows the ratio of deliveries (S/P), where S and P denote the total deliveries achieved by the Sequential and Parallel algorithms, respectively. Each plot varies one additional parameter: (a) the total number of delivery points, (b) the drone range, and (c) the drone velocity. Across all tested configurations, the ratio consistently remains above 1, indicating that the Sequential Greedy algorithm outperforms the Parallel Greedy approach.

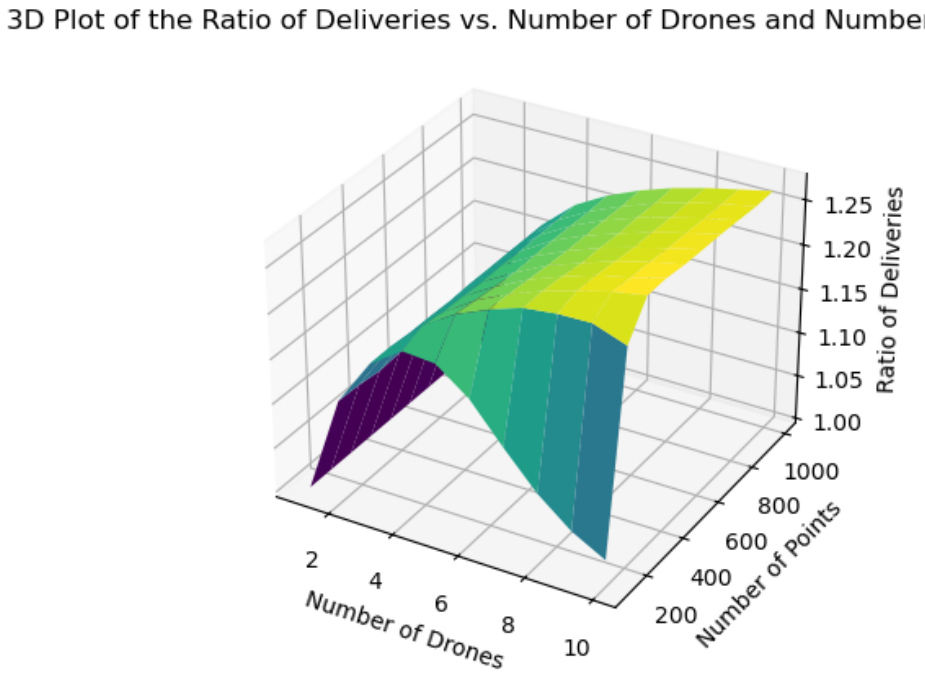


Figure 5: The Sequential Algorithm outperforms the Parallel Algorithm across all tested numbers of drones and number of delivery points.

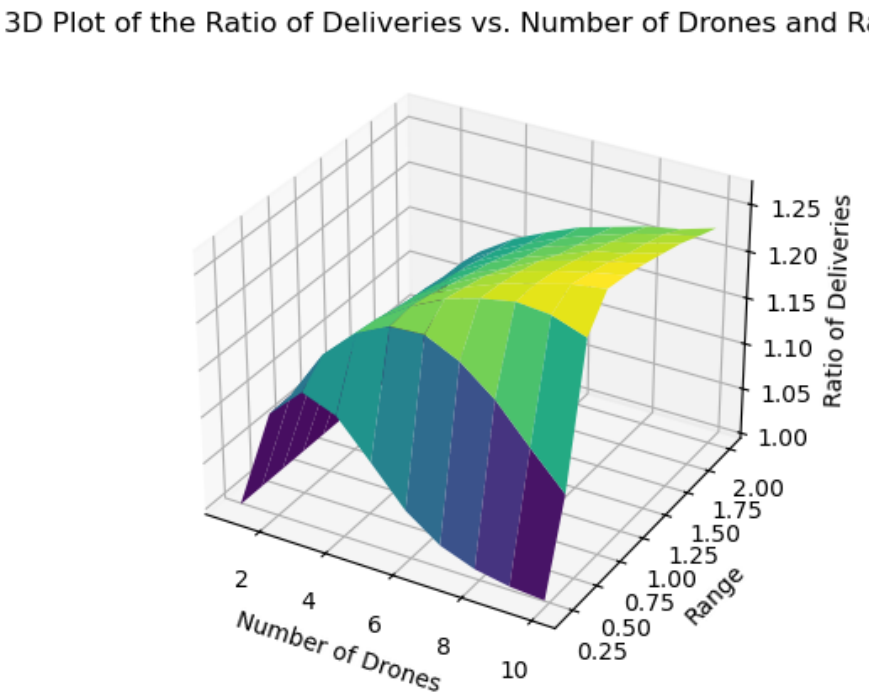


Figure 6: The Sequential Algorithm delivers more items than the Parallel Algorithm under every tested combination of drone count and effective range.

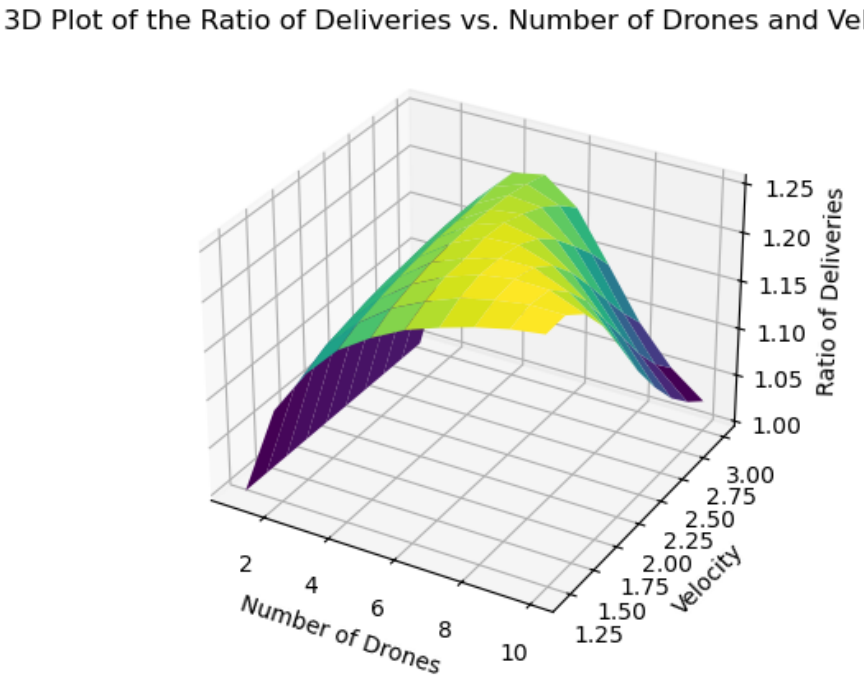


Figure 7: The Sequential Algorithm achieves a higher delivery count than the Parallel Algorithm across all variations in drone velocity and number of drones.

3. Execution Time Comparison

The execution time increases gradually with the number of drones for both algorithms. However, the Sequential Greedy Algorithm consistently performs faster than the Parallel Greedy Algorithm, as illustrated in Figure 8.

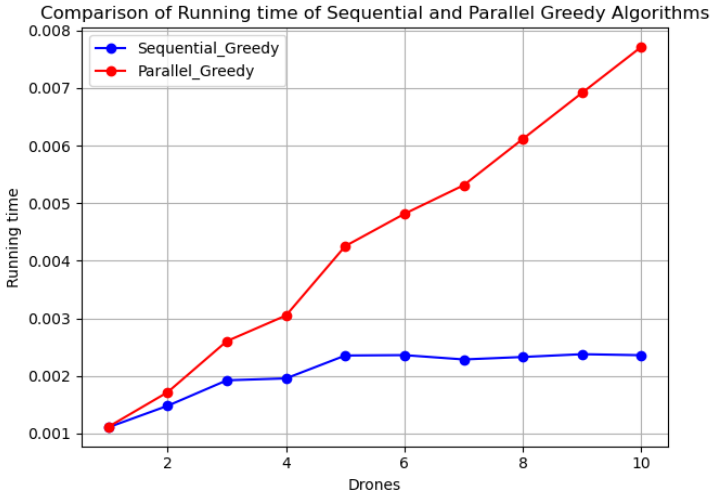


Figure 8: Sequential Greedy Algorithm has smaller execution time.

4. Drone Minimization

The goal of this experiment was to determine the minimum average number of drones required to deliver a given number of points (n) using two different greedy strategies. The results show a clear advantage for the Sequential Greedy algorithm across all data points. The Parallel Greedy algorithm requires significantly more drones to complete the same set of deliveries, as shown in Figure 9.

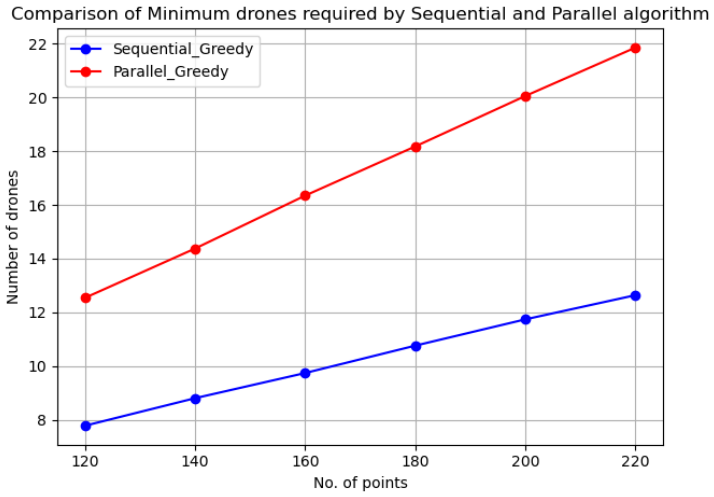


Figure 9: Sequential Greedy Algorithm requires lesser number of drones

Conclusion

The experiments demonstrate that the Sequential Greedy Algorithm consistently outperforms the Parallel Greedy Algorithm in multiple aspects of multi-drone delivery scheduling. Specifically:

- The Sequential Greedy approach achieves higher delivery counts across all configurations, with the advantage being most pronounced when the number of drones is small.

- The ratio analysis shows that the Sequential Greedy Algorithm outperforms the Parallel Greedy Algorithm (ratio ≥ 1.0) under all tested conditions across variations in the number of drones, number of points, velocity, and range.
- Execution time measurements indicate that the Sequential method is not only more efficient in deliveries but also faster, owing to its simpler scheduling logic compared to the Parallel approach, which requires additional coordination.
- In terms of drone minimization, Sequential requires fewer drones to complete the same set of deliveries, further emphasizing its resource efficiency.

Overall, these results suggest that for multi-drone delivery problems, a sequential scheduling strategy provides a more effective and computationally efficient solution compared to parallel scheduling.

This section considered general (random) point sets, where greedy choices are natural but structural guarantees are limited. In the next section, we restrict attention to proper instances, which impose geometric and interval-graph structure. This added structure enables dynamic-programming based scheduling while keeping the same coverage objective.

3 Truck-Drone Model with Restricted Set of Inputs: k-Drones

This section keeps the objective of maximizing the number of completed deliveries but changes the input setting. Instead of arbitrary random point sets, we assume the deliveries form a proper instance. The purpose of this restriction is algorithmic: it introduces structure that can be exploited by dynamic programming. We first formalize the proper-instance constraint, then develop DP-based multi-drone algorithms, and finally evaluate their scalability and solution quality.

3.1 Problem Formulation

This problem phase focuses on maximizing the number of deliveries made by k drones, specifically when the set of delivery points constitutes a proper instance. All physical constraints and scheduling parameters established in Section 2.1 (including truck velocity $v_T = 1$, drone velocity $v > 1$, range R, and the feasibility functions $es(d)$, $ls(d)$, $er(d)$, $lr(d)$ and $\mathrm{ret}(s, v, d)$) are fully applicable here. The fundamental objective remains to find a feasible multi-drone schedule S_{total} that maximizes the total number of deliveries.

3.1.1 The Proper Instance Constraint

The foundational work by Krizanc et al. [4] established that for the single-drone case, an optimal Dynamic Programming algorithm exists specifically for instances conforming to the following two conditions. The instance $I = (v, R, D)$ is considered a proper instance if the set of delivery points $D = \{d_1, d_2, \ldots, d_n\}$ adheres to:

1. For any two distinct delivery points d_i and d_j, d_j is not contained within the triangle formed by $[es(d_i), 0]$, d_i, and $[lr(d_i), 0]$. This condition ensures that delivery points have pairwise different x-coordinates and are not clustered in a narrow vertical band.

2. The set of intervals $\{[es(d_1), ls(d_1)], [es(d_2), ls(d_2)], \ldots, [es(d_n), ls(d_n)]\}$ forms a proper interval graph, meaning no interval is a subset of another.

Algorithm 3 is an implementation of the proper-instance constraints described in [4].

Algorithm 3: Generate Proper Instance of Points

Input: Number of points n, Range R, Velocity v, Maximum Truck Distance x_max
Output: Points belonging to the generated proper instance D

$D \leftarrow \emptyset$ $intervals \leftarrow \emptyset$
if $v \leq 1$ **then**
  return D
$y\_max \leftarrow \frac{R}{2v}\sqrt{v^2 - 1}$
while $|D| < n$ **do**
  Generate $x \sim U(0, x\_max)$
  Generate $y \sim U(0, y\_max)$
  if $x = 0$ ***or*** $y = 0$ **then**
    continue
  $d \leftarrow (x, y)$
  $(es, ls, lr) \leftarrow$ `calculate_es_ls_lr`(R, v, d)
  if (es, ls, lr) *is None* ***or*** $ls < 0$ **then**
    continue
  $valid \leftarrow$ True
  foreach $other\_d \in D$ **do**
    $(o\_es, o\_ls, o\_lr) \leftarrow$ `calculate_es_ls_lr`$(R, v, other\_d)$
    $A \leftarrow (o\_es, 0)$, $B \leftarrow other\_d$, $C \leftarrow (o\_lr, 0)$
    if `is_inside_triangle`(d, A, B, C) **then**
      $valid \leftarrow$ False
      break
  if $valid$ **then**
    $new\_interval \leftarrow (es, ls)$
    $is\_subset \leftarrow$ False
    foreach $inter \in intervals$ **do**
      if $(new\_interval[0] > inter[0]$ ***and*** $new\_interval[1] < inter[1])$ ***or***
       $(inter[0] > new\_interval[0]$ ***and*** $inter[1] < new\_interval[1])$ **then**
        $is\_subset \leftarrow$ True
        break
    if *not* is_subset **then**
      Append d to D
      Append $new\_interval$ to $intervals$
Sort D by x coordinate
return D

3.1.2 Optimal Dynamic Programming Algorithm

The foundational work by Krizanc et al. [4] introduced an optimal scheduling algorithm for the single-drone ($k = 1$) problem on any proper instance. This approach utilizes Dynamic Programming to find the maximum number of deliveries in $O(n^3)$ time, where n is the total number of delivery points. We restate it here because it is a subroutine and baseline for our multi-drone DP heuristics.

State Definition and Preprocessing The delivery points $D = \{d_1, d_2, \ldots, d_n\}$ are first sorted according to their x-coordinates. The algorithm defines a two-dimensional DP table, $T(i, j)$, with $O(n^2)$ entries:

- i: The number of deliveries performed - 1 ($1 \leq i \leq n$).
- j: The index of the last delivery point considered ($1 \leq j \leq n$).
- $T(i, j)$: The earliest delivery completion time (return time to the truck) for a schedule that performs exactly i deliveries from the subset $\{d_1, \ldots, d_j\}$, with the strict requirement that the i-th delivery must be d_j.

If a feasible schedule of length i ending at d_j is impossible, $T(i, j)$ is set to ∞.

Base Case and Recurrence Relation The table is filled using the following recurrence, which enforces feasibility and minimizes the completion time: This recurrence is enabled by the fact that the optimal schedule is monotone with respect to x-coordinates and possesses the optimal substructure property, a characteristic of proper instances proved in [4].

- **Base Case ($i = 0$):** The earliest completion time for a single mission to d_j. The launch time s must be $\max(0, es(d_j))$, ensuring the truck is past the origin and within the point's earliest launch time.
$$T(0, j) = \text{ret}(\max(0, es(d_j)), v, d_j)$$
- **Recurrence ($i \geq 1$):** The optimal substructure property dictates that the schedule with the earliest completion time for i deliveries ending at d_j is found by extending the optimal schedule of length $i - 1$ ending at some feasible preceding point $d_{j'}$ ($j' < j$). The launch time s for the i-th mission is the maximum of the point's earliest launch time $es(d_j)$ and the return time of the $(i - 1)$-th mission, $T(i - 1, j')$.
$$T(i, j) = \min_{j' < j} \{\text{ret}\left(\max(es(d_j), T(i - 1, j')), v, d_j\right)\}$$
The $\text{ret}(s, v, d_j)$ function correctly calculates the return time after servicing d_j, assuming the launch occurs at time s. The minimization selects the predecessor that allows the i-th mission to d_j to finish earliest, subject to the launch time s being less than or equal to $ls(d_j)$. The full pseudocode for computing the DP table is presented in Algorithm 4.

Algorithm 4: compute_T_Table($D, R, v, x_{\max}$)

Input: Ordered deliveries D, Range R, velocity v, and Max. Truck Distance $x_{\max}$
Output: Predecessor table prev

```
n ← |D|
T ← array of n entries initialized to ∞
T_prev ← array of n entries initialized to ∞
prev ← n × n matrix initialized to −1
s ← 0
Compute es_ls_table[j] ← calculate_es_ls(R, v, D[j]) for all j
// --- Initialization layer i = 0 ---
for j ← 0 to n − 1 do
    (es, ls) ← es_ls_table[j]
    if (es, ls) is valid then
        if s ≤ ls then
            st ← max(s, es)
            T[j] ← ret(st, v, D[j])
            prev[0][j] ← 0
T_prev ← T
Reset T to all ∞
// --- Dynamic layers i = 1 to n − 1 ---
for i ← 1 to n − 1 do
    flag ← False
    for j ← i to n − 1 do
        (es, ls) ← es_ls_table[j]
        if (es, ls) is invalid then
            continue
        for j′ ← 0 to j − 1 do
            s ← T_prev[j′]
            if s ≤ ls and s < x_max then
                s ← max(s, es)
                cand ← ret(s, v, D[j])
                if cand < T[j] then
                    T[j] ← cand
                    prev[i][j] ← j′
                    flag ← True
    if not flag then
        break                          // No more feasible extensions
    T_prev ← T
    Reset T to all ∞
return prev
```

Solution and Runtime The largest i with any finite $T(i, j)$ gives the maximum deliveries, and the schedule is reconstructed by backtracking as in Algorithm 5.

Algorithm 5: calc_Schedule(prev)

Input: Predecessor table prev of size $n \times n$
Output: Sequence of selected delivery indices
$n \leftarrow |\text{prev}|$
$S \leftarrow$ empty list
`// Find deepest valid state` (i, j)
Function `search()`**:**
    for $i \leftarrow n-1$ **to** 0 **do**
        for $j \leftarrow 0$ **to** $n-1$ **do**
            if *prev*$[i][j] \neq -1$ **then**
                return (i, j)
    return $(-1, -1)$
$(i, j) \leftarrow$ `search()`
if $i = -1$ **then**
    return S `// No feasible schedule`
`// Backtrack predecessors`
while $i \geq 0$ **do**
    Append j to S
    $j \leftarrow \text{prev}[i][j]$
    $i \leftarrow i - 1$
Reverse S
return S

The running time is dominated by the table computation. The $O(n^2)$ table entries, combined with the $O(n)$ time required to check all predecessors j' for each entry, yields the final complexity:

$$\text{Runtime} = O(n^2) \cdot O(n) = O(n^3)$$

Our objective is to develop efficient algorithms that optimally leverage the k available drones within these stringent constraints. We developed four distinct algorithmic strategies to address this objective: the Sequential 1-DP, Parallel DP, the Sequential 2-DP and the Sequential 2-DP*.

3.2 Algorithms and Analysis

Common Inputs
The algorithms share a set of common inputs:

- **deliveries**: A list of delivery points in the form $[(x_1, y_1), (x_2, y_2), \dots]$.
- **R**: The maximum range the drone can travel on a single trip.
- **v**: The velocity of the drone (v_d).
- **k**: Number of the drones.
- **max_truck_distance**: The maximum endpoint of the route, representing the farthest point the truck must reach.

3.2.1 Sequential 1-DP Algorithm

Algorithm Description The Sequential 1-DP algorithm is a greedy heuristic designed to tackle the k-drones delivery problem by maximizing the number of deliveries in each step. This algorithm is new in this thesis. Its core strategy is to leverage the optimal $O(n^3)$ single-drone Dynamic Programming (DP) algorithm from [4] as a subroutine and applies it greedily across drones. The Sequential 1-DP algorithm iterates exactly k times (once for each drone). In every iteration, the remaining available, unscheduled points are passed to the DP algorithm, which finds the single longest possible feasible delivery route from that current set. This best schedule is immediately assigned to the current drone, and all included delivered points are then permanently removed from the available pool for future iterations. This sequential, greedy process ensures that each drone obtains the single best, locally optimal schedule from the remaining workload. The complete pseudocode for the Sequential 1-DP algorithm is presented in Algorithm 6. The running time is dominated by the k calls to the $O(n^3)$ DP algorithm. Since the number of points in each iteration decreases but remains $O(n)$ in the worst case, the total complexity is $O(k \cdot n^3)$. While each drone's schedule is individually optimal for the points it serves, the Sequential 1-DP is a greedy heuristic. The sequence in which points are removed can affect the overall solution, meaning the final total number of deliveries is not guaranteed to be globally optimal for the k-drones problem.

Algorithm 6: Sequential_1DP($points, R, v, k, x_{\max}$)

Input: Set of delivery points, parameters R, v, number of drones k, Maximum Truck Distance $x_{\max}$
Output: List of k (or fewer) schedules
$scheds \leftarrow$ empty list
`// Sort points by earliest-start coordinate`
$idx\_pts \leftarrow$ list of pairs $(i, points[i])$
Sort idx_pts by delivery coordinate
$s\_pts \leftarrow$ sorted delivery points
$s\_idx \leftarrow$ original indices in sorted order
$avail \leftarrow [0, 1, \ldots, |s\_pts| - 1]$
for $iter \leftarrow 1$ **to** k **do**
  if $avail$ *is empty* **then**
    break `// No tasks left`
  $cur \leftarrow [\, s\_pts[i] \mid i \in avail \,]$
  prev $\leftarrow$ compute_T_Table($cur, R, v, x_{\max}$)
  $rel \leftarrow$ calc_Schedule(prev)
  if rel *is empty* **then**
    break `// No feasible schedule for next drone`
  `// Map relative indices → sorted-space indices`
  $abs\_sorted \leftarrow [\, avail[i] \mid i \in rel \,]$
  `// Map sorted indices back to original global indices`
  $orig \leftarrow [\, s\_idx[j] \mid j \in abs\_sorted \,]$
  Append $orig$ to $scheds$
  `// Remove scheduled deliveries from availability`
  $used \leftarrow$ set(abs_sorted)
  $avail \leftarrow [\, i \in avail \mid i \notin used \,]$
return $scheds$

Approximation Ratio Analysis for Sequential Dynamic Programming Algorithm

This bound follows the same coverage-style argument as in classical greedy coverage analysis; we include it to quantify the loss from sequentially selecting optimal single-drone schedules on remaining points.

Useful Parameters:

- V_0: The set of all delivery points.
- k: Number of drones.
- DP_i: Set of points delivered by the i-th drone in the sequential DP schedule.
- OPT_i: Set of points delivered by the i-th drone in the optimal schedule.
- $DP(k)$: Total number of unique points delivered by k drones in the sequential DP schedule.
- $OPT(k)$: Total number of unique points delivered by k drones in the optimal schedules.
- V_{i-1}: The set of remaining unserved delivery points at the start of iteration i, defined as the initial set V_0 excluding all points served by the previous $i-1$ DP drone schedules:

$$V_{i-1} = V_0 \setminus \bigcup_{j=1}^{i-1} DP_j.$$

Observation (Disjointness and Total Value).
Since each delivery point can be assigned to at most one drone, all individual drone schedules are disjoint:

$$DP_i \cap DP_j = \emptyset \quad \text{and} \quad OPT_i \cap OPT_j = \emptyset \quad \forall i \neq j.$$

Consequently, the total value of any solution is the sum of its disjoint schedule sizes:

$$DP(k) = \sum_{i=1}^{k} |DP_i|, \qquad OPT(k) = \sum_{i=1}^{k} |OPT_i|.$$

Lemma 5 (Lower Bound for DP_1). Let $OPT_1, \ldots, OPT_k$ be the k disjoint optimal drone schedules, and let $|OPT_{\max}| = \max_i |OPT_i|$ denote the largest single schedule.
By definition, the optimal single schedule computed by the DP algorithm on the full set of delivery points satisfies

$$OPT(1) \geq |OPT_{\max}|,$$

because the DP solution can always reproduce or exceed the largest among the k disjoint optimal schedules.
Furthermore, since the largest optimal schedule is at least as large as the average of all k schedules, we have

$$OPT(1) \geq |OPT_{\max}| \geq \frac{1}{k} \sum_{i=1}^{k} |OPT_i| = \frac{1}{k} OPT(k).$$

Lemma 6 (Local Optimality Property).
At each iteration i, the DP algorithm selects the drone schedule DP_i that maximizes the number

of remaining deliverable points from V_{i-1}. Let $OPT_1(V_{i-1})$ denote the optimal single drone schedule restricted to the remaining set V_{i-1}. Since DP finds the exact optimum:

$$|DP_i| = |OPT_1(V_{i-1})|.$$

This holds for all $i = 1, \ldots, k$.

Theorem 3.1. *Let $DP(k)$ denote the total number of deliveries achieved by the sequential dynamic programming algorithm using k drones, and let $OPT(k)$ denote the size of an optimal set of k disjoint schedules. Then we have*

$$DP(k) \geq OPT(k) \left(1 - \left(1 - \frac{1}{k}\right)^k\right).$$

Proof. **Bound for the first schedule.**
From the Lower Bound for DP_1 in Lemma 5, the first DP schedule satisfies

$$|DP_1| = OPT_1(V_0) \geq \frac{1}{k} OPT(k),$$

where $OPT_1(V_0)$ denotes the optimal single schedule on the full set of points.

Bounds for subsequent schedules.
Fix an iteration $i \in \{2, \ldots, k\}$. Let

$$V_{i-1} = V_0 \setminus \bigcup_{j=1}^{i-1} DP_j$$

be the remaining points after the first $i-1$ DP selections. By Lemma 6 the DP choice satisfies

$$|DP_i| = |OPT_1(V_{i-1})|,$$

where $OPT_1(V_{i-1})$ denotes the optimal single schedule on V_{i-1}.

To relate $|OPT_1(V_{i-1})|$ to $OPT(k)$, consider the k disjoint optimal schedules $OPT_1, \ldots, OPT_k$. After the first $i-1$ DP schedules, the number of points in the remaining portions of the optimal schedules satisfies

$$\sum_{\ell=1}^{k} |OPT_\ell \setminus \bigcup_{j=1}^{i-1} DP_j| = \sum_{\ell=1}^{k} |OPT_\ell| - \sum_{\ell=1}^{k} |OPT_\ell \cap \bigcup_{j=1}^{i-1} DP_j|.$$

Since the DP schedules DP_j are disjoint, we have

$$\sum_{\ell=1}^{k} |OPT_\ell \cap \bigcup_{j=1}^{i-1} DP_j| \leq \sum_{j=1}^{i-1} |DP_j|.$$

Combining these gives the key inequality:

$$\sum_{\ell=1}^{k} |OPT_\ell \setminus \bigcup_{j=1}^{i-1} DP_j| \geq OPT(k) - \sum_{j=1}^{i-1} |DP_j|.$$

Since the value of the largest of these k disjoint subsets must be at least as large as their average, we have:

$$\max_{\ell} |OPT_\ell \setminus \bigcup_{j=1}^{i-1} DP_j| \geq \frac{1}{k}\Big(OPT(k) - \sum_{j=1}^{i-1} |DP_j|\Big).$$

The DP schedule on the remaining points, $OPT_1(V_{i-1})$, achieves this maximum exactly. Hence

$$|DP_i| = |OPT_1(V_{i-1})| \geq \frac{1}{k}\Big(OPT(k) - \sum_{j=1}^{i-1} |DP_j|\Big),$$

and in particular:

$$|DP_2| \geq \frac{1}{k}\big(OPT(k) - |DP_1|\big), \qquad |DP_3| \geq \frac{1}{k}\big(OPT(k) - (|DP_1| + |DP_2|)\big),$$

and so on, up to

$$|DP_k| \geq \frac{1}{k}\left(OPT(k) - \sum_{j=1}^{k-1} |DP_j|\right).$$

Iterative expansion.
Adding the inequalities iteratively, we obtain

$$\begin{aligned}
|DP_1| + |DP_2| &\geq \frac{1}{k}\Big(OPT(k) + (k-1)|DP_1|\Big), \\
|DP_1| + |DP_2| + |DP_3| &\geq \frac{1}{k}\Big(OPT(k) + (k-1)(|DP_1| + |DP_2|)\Big), \\
&\vdots \\
DP(k) = \sum_{i=1}^{k} |DP_i| &\geq \frac{1}{k}\left(OPT(k) + (k-1)\sum_{i=1}^{k-1} |DP_i|\right),
\end{aligned}$$

where the final step follows from the Observation, which ensures that the DP schedules DP_i are disjoint and additive.

Closed-form expression. Expanding this recursively leads to the geometric series

$$DP(k) \geq \frac{1}{k} OPT(k) \left[1 + \left(1 - \frac{1}{k}\right) + \left(1 - \frac{1}{k}\right)^2 + \cdots + \left(1 - \frac{1}{k}\right)^{k-1}\right].$$

Summing the series gives the generalized approximation ratio:

$$DP(k) \geq \frac{1}{k} OPT(k) \cdot \frac{1 - \left(1 - \frac{1}{k}\right)^k}{1 - \left(1 - \frac{1}{k}\right)} = OPT(k)\left(1 - \left(1 - \frac{1}{k}\right)^k\right).$$

□

Remark. This proof mirrors the classical analysis of the greedy algorithm for the Maximum Coverage problem. In the Maximum Coverage context, the greedy choice selects a set that covers a maximum fraction of the remaining uncovered elements. Here, each sequential schedule DP_i acts similarly, exploiting the fact that the best single schedule on the remaining points

$(OPT_1(V_{i-1}))$ must capture at least a $\frac{1}{k}$ fraction of the total remaining optimal coverage. The Disjointness Constraint (each point served once) is crucial, ensuring both the greedy and optimal schedules form partitions of their served points, which allows for the exact calculation of the total approximation bound $1 - \left(1 - \frac{1}{k}\right)^k$.

Asymptotic Behavior When k tends to infinity, we use the limit $\lim_{n\to\infty} \left(1 + \frac{x}{n}\right)^n = e^x$. Here, let $n = k$ and $x = -1$.

$$\lim_{k\to\infty} \left(1 - \frac{1}{k}\right)^k = \lim_{k\to\infty} \left(\left(1 + \frac{-1}{k}\right)^k\right) = e^{-1}$$

So, the approximation factor approaches:

$$\lim_{k\to\infty} \left[1 - \left(1 - \frac{1}{k}\right)^k\right] = 1 - e^{-1} \approx 1 - 0.3679 \approx 0.6321.$$

This asymptotic behavior is illustrated by the curve shown in Figure 10.

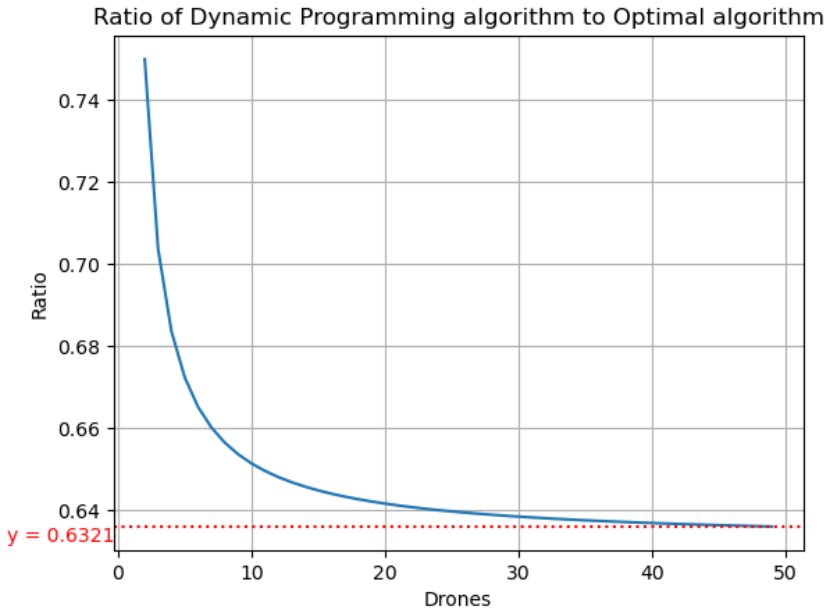


Figure 10: Approximation factor $DP(k)/OPT(k)$ as $k \to \infty$.

3.2.2 Parallel DP Algorithm

Algorithm Description Parallel DP is proposed in this thesis as a multi-drone dynamic programming heuristic for proper instances. Parallel_DP schedules a set of n delivery points D among k drones using a dynamic programming table $T[i][j][d]$, which represents the completion time of drone d after serving the $(i+1)$-th assigned point ending at location j. The algorithm operates in two phases. In *Phase 1* $(0 \leq i < k)$, the first k drones are initialized by assigning point j to drone i, while for each drone $d < i$ the minimum prior completion times from all feasible predecessor points $j' < j$ in $T[i-1]$ are copied into $T[i][j][d]$ to preserve consistency across drones. In *Phase 2* $(k \leq i < n)$, each point j selects an optimal predecessor (j', d') from $T[i-1]$ that minimizes the achievable arrival time, after which point j is assigned to drone d' and $T[i][j][d']$ is updated accordingly. For all remaining drones $d \neq d'$, the state

values are copied directly from the predecessor row $T[i-1][j']$ to maintain synchronization across the DP layers. The complete pseudocode for constructing the dynamic programming table in the Parallel_DP algorithm is presented in Algorithm 7. After the DP table is fully constructed, the procedure *Calc_Schedule_Parallel* backtracks through the stored predecessor table `prev` to reconstruct the k drone routes in reverse order, yielding the final parallel schedule. The procedure used to reconstruct the k parallel drone routes from the DP predecessor table is shown in Algorithm 8.

The running time of the `Parallel_DP` algorithm is dominated by the construction of the dynamic programming table T in `compute_T_Table_Parallel`. Let n be the number of delivery points and k the number of drones. In Phase 1 ($0 \leq i < k$), initializing the first k points requires $O(k^2n^2)$ time due to the search for minimum prior completion times for each drone $d < i$ over all feasible predecessor points. In Phase 2 ($k \leq i < n$), extending the schedules requires selecting the optimal predecessor (j^*, d^*) for each state (i, j) by checking all k drones and all preceding points j', giving $O(kn)$ per state. Summing over the $O(n^2)$ states, Phase 2 dominates with $O(kn^3)$ complexity. Schedule reconstruction (`Calc_Schedule_Parallel`) runs in $O(n^2)$ and is negligible, yielding an overall time complexity of $O(kn^3)$. We analyze runtime and evaluate empirically; we do not claim a formal approximation guarantee for this heuristic.

DP Formulation for Parallel_DP Algorithm

State The DP state $T_{i,j,d}$ represents the completion time of drone d when exactly $i+1$ points have been served, and point j is the $(i+1)^{\text{th}}$ point served overall. Here, N is the total number of points, k is the number of drones, and $x_{\max}$ is the maximum travel distance of the truck.

Initialization ($0 \leq i < k$) This phase establishes the base schedules for the first k drones. The index i represents the total number of points served minus one.

$$T[i][j][d] = \begin{cases} \text{ret}(\max(0, es(j)), v, D[j]), & \text{if } d = i \text{ (Drone } i \text{ serves its first point } j) \\ \min_{j'<j} \{T[i-1][j'][d]\}, & \text{if } d < i \text{ (Time copied from best predecessor } j') \\ 0, & \text{if } d > i \text{ (Drone not yet initialized)} \end{cases}$$

Recurrence (Extension, $k \leq i < N$) This phase assigns the new point j to the drone that can serve it earliest.

The algorithm finds the optimal predecessor state (j^*, d^*) from the set of i served points (ending at j') that minimizes the predecessor's completion time, $T[i-1][j'][d']$, while ensuring feasibility:

$$(j^*, d^*) = \underset{d' \in [0,K-1],\, j' \in [i-1, j-1]}{\text{argmin}} \{T[i-1][j'][d'] \mid T[i-1][j'][d'] \leq ls(j) \text{ and } T[i-1][j'][d'] < x_{\max}\}$$

The new state $T[i][j][d]$ is updated based on the optimal choice (j^*, d^*):

$$T[i][j][d] = \begin{cases} \text{ret}(\max(es(j), T[i-1][j^*][d^*]), v, D[j]), & d = d^*, \\ T[i-1][j^*][d], & d \neq d^*. \end{cases}$$

(Drone d^* serves point j; other drones copy predecessor times.)

Algorithm 7: compute_T_Table_Parallel($D, R, v, k, x_{\max}$)

Input: D: delivery points; R: Range; v: velocity; k: drones; $x_{\max}$: max truck distance
Output: $prev$: predecessor table; $I[k-1]$: index table for initialization
$n \leftarrow |D|$
Initialize $T[i][j][d] \leftarrow \infty$, $I[i][j][d] \leftarrow -1$, $prev[i][j] \leftarrow (-1, -1)$

Phase 1: $0 \leq i < k$
for $i \leftarrow 0$ **to** $k-1$ **do**
  for $j \leftarrow i$ **to** $n-1$ **do**
    $set \leftarrow \emptyset$
    for $d \leftarrow 0$ **to** $i-1$ **do**
      $prev\_time \leftarrow \infty$
      for $jp \leftarrow 0$ **to** $j-1$ **do**
        if $T[i-1][jp][d] < prev\_time$ ***and*** $I[i-1][jp][d] \notin set$ **then**
          $prev\_time \leftarrow T[i-1][jp][d]$
          $j' \leftarrow jp$
      $T[i][j][d] \leftarrow prev\_time$
      $I[i][j][d] \leftarrow I[i-1][j'][d]$
      $set \leftarrow set \cup \{I[i][j][d]\}$
    $(es, ls) \leftarrow$ calculate_es_ls$(R, v, D[j])$
    $T[i][j][i] \leftarrow$ ret$(\max(0, es), v, D[j])$
    $I[i][j][i] \leftarrow j$
    for $d \leftarrow i+1$ **to** $k-1$ **do**
      $T[i][j][d] \leftarrow 0$

Phase 2: $i \geq k$
for $i \leftarrow k$ **to** $n-1$ **do**
  for $j \leftarrow i$ **to** $n-1$ **do**
    $prev\_time \leftarrow \infty$
    $(es, ls) \leftarrow$ calculate_es_ls$(R, v, D[j])$
    for $d \leftarrow 0$ **to** $k-1$ **do**
      for $jp \leftarrow i-1$ **to** $j-1$ **do**
        if $T[i-1][jp][d] < prev\_time$
        and $T[i-1][jp][d] < x_{\max}$
        and $T[i-1][jp][d] \leq ls$ **then**
          $prev\_time \leftarrow T[i-1][jp][d]$
          $prev[i][j] \leftarrow (jp, d)$
    if $prev\_time \neq \infty$ **then**
      $(j', d^*) \leftarrow prev[i][j]$
      $T[i][j][d^*] \leftarrow$ ret$(\max(es, prev_time), v, D[j])$
      for $d \leftarrow 0$ **to** $k-1$ **do**
        if $d \neq d^*$ **then**
          $T[i][j][d] \leftarrow T[i-1][j'][d]$

return $(prev, I[k-1])$

Algorithm 8: Calc_Schedule_Parallel($I_{\text{last}}, prev, k$)

Input: I_{last}: final index table; $prev$: predecessor table; k: number of drones
Output: A list of k schedules
$n \leftarrow$ length of $prev$;

Function Search():
    for $i \leftarrow n-1$ **to** 0 **do**
        for $j \leftarrow 0$ **to** $n-1$ **do**
            if $prev[i][j] \neq (-1,-1)$ **then**
                return (i,j)
    return $(-1,-1)$;

$(i,j) \leftarrow$ Search() ;

Initialize $schedules[d] \leftarrow$ empty list for each $d = 0, \ldots, k-1$;

while $i \geq k$ **do**
    $(j', d) \leftarrow prev[i][j]$;
    Append j to $schedules[d]$;
    $j \leftarrow j'$;
    $i \leftarrow i-1$;

for $d \leftarrow 0$ **to** $k-1$ **do**
    Append $I_{\text{last}}[j][d]$ to $schedules[d]$;

for $d \leftarrow 0$ **to** $k-1$ **do**
    Reverse $schedules[d]$;

return $schedules$;

3.2.3 Sequential 2-DP Algorithm

Algorithm Description Sequential_2DP is an original heuristic DP construction introduced in this thesis to exploit paired-drone scheduling structure. The Sequential_2DP algorithm addresses the time-constrained routing problem for k drones by iteratively computing locally optimal dual paths for pairs of drones until all delivery points are scheduled. Its core component is the three-dimensional dynamic programming (3D DP) procedure compute_T_Table_2DP, which constructs a locally optimal pair of routes for two drones over a set of N available points.

The DP maintains a state

$$T[I][i][j] = (t_1, t_2),$$

representing the completion times when Drone 1 ends at point i and Drone 2 ends at point j after $2(I+1)$ points have been assigned. Here, the index I counts the number of dual-assignment iterations.

In the base case ($I = 0$), $T[0][i][j]$ is initialized by computing the earliest feasible completion times for points i and j. In the recurrent case ($I > 0$), the DP transitions from a predecessor state $T[I-1][i_1][j_1]$ to $T[I][i][j]$ by selecting the predecessor that minimizes the resulting makespan $\max(t_1, t_2)$, while ensuring feasibility with respect to time windows and previously assigned points. Unlike Sequential_2DP*, only a single predecessor is retained for each state, resulting in a greedy commitment to the locally optimal dual extension.

The function calc_Schedule_2DP backtracks through the predecessor table to reconstruct

the two selected routes. The main routine Sequential_2DP repeatedly invokes this DP on the current set of available points, extracts the resulting paths, removes the scheduled points, and stores the schedules in terms of original indices. This process continues until all points are scheduled or $\lfloor k/2 \rfloor$ paired routes are obtained, with an optional final single-drone route if k is odd. We analyze runtime and evaluate empirically; we do not claim a formal approximation guarantee for this heuristic.

The running time of compute_T_Table_2DP for N points is $O(N^5)$, due to four nested loops over (i, j, i_1, j_1) within an outer loop over $I = O(N)$. Since Sequential_2DP invokes this DP repeatedly on a decreasing number of points, the overall worst-case time complexity for an initial set of N points is

$$\sum_{i=1}^{\lfloor k/2 \rfloor} O(N^5) = O(kN^5).$$

DP Formulation for Sequential_2DP

State

The DP state stores the completion times of Drone 1 and Drone 2 after $2(I+1)$ points have been assigned:

$$T[I][i][j] = (t_1, t_2),$$

where point i is served by Drone 1 and point j is served by Drone 2 in iteration I. The predecessor pointer is

$$\text{prev}[I][i][j] = (i_1, j_1).$$

Base Case ($I = 0$)

$$T[0][i][j] = \begin{cases} (\text{ret}(\max(0, es(i)), v, D[i]),\ \text{ret}(\max(0, es(j)), v, D[j])), & \text{if } i \neq j,\ ls(i) \geq 0,\ ls(j) \geq 0, \\ (\infty, \infty), & \text{otherwise.} \end{cases}$$

$$\text{prev}[0][i][j] = (-1, -1).$$

Recurrent Case ($I > 0$)

For each pair (i, j), the DP considers predecessor states (i_1, j_1) satisfying

$$(t_1, t_2) = T[I-1][i_1][j_1],$$

$$t_1 \leq ls(i), \qquad t_2 \leq ls(j), \qquad \{i, j\} \cap \text{UsedPoints}[I-1][i_1][j_1] = \emptyset.$$

Among all such predecessors, the DP greedily selects the one minimizing the resulting makespan:

$$(i_1^*, j_1^*) = \underset{i_1 < i,\ j_1 < j}{\arg\min}\ \max(\text{ret}(\max(t_1, es(i)), v, D[i]),\ \text{ret}(\max(t_2, es(j)), v, D[j])).$$

The recurrence is

$$T[I][i][j] = (\text{ret}(\max(t_1^*, es(i)), v, D[i]),\ \text{ret}(\max(t_2^*, es(j)), v, D[j])),$$

where $(t_1^*, t_2^*) = T[I-1][i_1^*][j_1^*]$, and

$$\text{prev}[I][i][j] = (i_1^*, j_1^*).$$

Algorithm 9: `compute_T_Table_2DP`$(D, R, v, x_{\max})$

Input : D = delivery locations, R = Range, v = velocity, $x_{\max}$ = Maximum Truck Distance

Output: Table `prev` storing predecessor states

$N \leftarrow |D|$, rows $\leftarrow N/2$
Initialize $T[i][j] \leftarrow (\infty, \infty)$, $T_{\text{prev}}[i][j] \leftarrow (\infty, \infty)$ for all i, j
Initialize prev$[I][i][j] \leftarrow (-1, -1)$, usedSets$[I][i][j] \leftarrow \emptyset$
Precompute $(es[d], ls[d])$ for all $d \in D$
for $I \leftarrow 0$ **to** *rows* $- 1$ **do**
  flag $\leftarrow$ False
  for $i \leftarrow 0$ **to** $N - 1$ **do**
    for $j \leftarrow 0$ **to** $N - 1$ **do**
      if $i = j$ **then**
        continue
      if $I = 0$ **then**
        Compute earliest feasible times r_1, r_2 for points i, j
        if $r_1 < \infty$ *and* $r_2 < \infty$ **then**
          $T[i][j] \leftarrow (r_1, r_2)$
          prev$[I][i][j] \leftarrow (0, 0)$
          usedSets$[I][i][j] \leftarrow \{i, j\}$
          flag $\leftarrow$ True
      else
        minVal $\leftarrow \infty$
        for $i_1 \leftarrow 0$ **to** $i - 1$ **do**
          for $j_1 \leftarrow 0$ **to** $j - 1$ **do**
            if $T_{prev}[i_1][j_1] = (\infty, \infty)$ **then**
              continue
            $\mathcal{S} \leftarrow$ usedSets$[I - 1][i_1][j_1]$
            if $i \in \mathcal{S}$ *or* $j \in \mathcal{S}$ **then**
              continue
            Compute feasible arrival times r_1, r_2 from state (i_1, j_1)
            makespan $\leftarrow \max(r_1, r_2)$
            if $r_1 < \infty$ *and* $r_2 < \infty$ *and makespan* $<$ *minVal* **then**
              $T[i][j] \leftarrow (r_1, r_2)$
              prev$[I][i][j] \leftarrow (i_1, j_1)$
              usedSets$[I][i][j] \leftarrow \mathcal{S} \cup \{i, j\}$
              minVal $\leftarrow$ makespan
              flag $\leftarrow$ True
  if *flag* $=$ *False* **then**
    break
  $T_{\text{prev}} \leftarrow T$
  Reset T to (∞, ∞) for all entries
return *prev*

Algorithm 10: `calc_Schedule_2DP(prev)`

Input : DP predecessor table `prev`
Output: Two-Drone schedule schedule$\{0\}$, schedule$\{1\}$
Initialize two empty routes schedule$\{0\}$ and schedule$\{1\}$
Find the last non-$(-1,-1)$ entry in `prev` and store it in (I^*, i^*, j^*)
if $I^* = -1$ **then**
  return *schedule*
Append i^* to schedule$\{0\}$ and append j^* to schedule$\{1\}$
while $I^* > 0$ **do**
  $(i^*, j^*) \leftarrow$ prev$[I^*][i^*][j^*]$
  Append i^* to schedule$\{0\}$ and j^* to schedule$\{1\}$
  $I^* \leftarrow I^* - 1$
Reverse both schedules
return *schedule*

Algorithm 11: `Sequential_2DP`$(P, R, v, k, x_{\max})$

Input : P = points, R = Range, v = speed, k = number of drones, $x_{\max}$ = Maximum Truck Distance
Output: A set of k schedules
Sort points by x-coordinate; store original indices
avail $\leftarrow$ indices of all points
schedules $\leftarrow \emptyset$
for $u \leftarrow 1$ **to** $k/2$ **do**
  $D \leftarrow$ points indexed by avail
  prev $\leftarrow$ `compute_T_Table_2DP`$(D, R, v, x_{\max})$
  routes $\leftarrow$ `calc_Schedule_2DP`(prev)
  Convert local indices to global via sorted mapping
  Append both routes to schedules
  Remove used points from avail
if k *is odd and avail not empty* **then**
  Solve remaining points with 1-DP
  Append resulting route to schedules
return *schedules*

3.2.4 Sequential 2-DP* Algorithm

The Sequential_2DP* algorithm is a refinement of Sequential_2DP designed to address the limitations of greedy commitment in dual-drone scheduling. Unlike Sequential_2DP, which selects a single locally optimal dual route at each step, Sequential_2DP* preserves all feasible extensions within a two-drone dynamic programming step and selects option minimizing the makespan. It solves the time-constrained routing problem for a set of N points using a sequential partitioning strategy optimized for pairs of drones.

The core of the algorithm is a three-dimensional dynamic program, compute_T_Table_2DP*, which constructs an optimal dual schedule over the remaining set of points. Each DP state $T[I][i][j]$ stores the pair of completion times (t_1, t_2) after assigning $I+1$ total points, with Drone 1 ending at point i and Drone 2 ending at point j. Here, the index I counts the total number of points assigned across both drones; extending one drone increases I by one, while extending both drones simultaneously increases I by two.

The recurrence supports three transition types: extending only Drone 1 from a prior state $T[I-1][i_1][j]$, extending only Drone 2 from a state $T[I-1][i][j_1]$, or extending both drones simultaneously from $T[I-2][i_1][j_1]$. For each transition, the algorithm computes candidate completion times, ensures that newly added points have not been previously visited (tracked via `used_sets`), verifies time-window feasibility, and retains the transition that minimizes the makespan $\max(t_1, t_2)$. We analyze runtime and evaluate empirically; we do not claim a formal approximation guarantee for this heuristic.

The main routine Sequential_2DP* repeatedly invokes the DP, extracts the optimal pair of routes using calc_Schedule_2DP*, removes the selected points from further consideration, and iterates until all $\lfloor K/2 \rfloor$ paired schedules are produced or no points remain. If K is odd, a final single-drone schedule is generated using the one-dimensional DP compute_T_Table. The overall worst-case running time is $O(kN^5)$, dominated by the $O(N^5)$ complexity of compute_T_Table_2DP* applied to successively smaller problem sizes.

DP Formulation for Sequential_2DP*

State Definition
The DP state records the completion times of Drone 1 and Drone 2 after $I+1$ total point assignments:

$$T[I][i][j] = (\hat{t}_1, \hat{t}_2),$$

where $\hat{t}_1$ is the completion time when Drone 1's last served point is i, and $\hat{t}_2$ is the completion time when Drone 2's last served point is j. The predecessor pointer is

$$\text{prev}[I][i][j] = (i_1, j_1).$$

Base Case (Initialization, $I = 0$)
The base case assigns one distinct point to each drone, provided both deliveries are individually feasible:

$$T[0][i][j] = \begin{cases} (\text{ret}(\max(0, es(i)), v, D[i]),\ \text{ret}(\max(0, es(j)), v, D[j])), & \text{if valid,} \\ (\infty, \infty), & \text{otherwise.} \end{cases}$$

$$\text{where valid means } i \neq j,\ ls(i) \geq 0,\ ls(j) \geq 0.$$

Recurrence (Extension, $I > 0$)
For $I > 0$, the optimal state $T[I][i][j]$ is obtained by minimizing the makespan $\max(\hat{t}_1, \hat{t}_2)$ over all feasible transition results:

$$T[I][i][j] = \underset{(\hat{t}_1, \hat{t}_2) \in C(i,j)}{\arg\min} \left\{ \max(\hat{t}_1, \hat{t}_2) \right\}.$$

The candidate set $C(i, j)$ is formed by the union of the following transitions:

1. **Transition A: Drone 1 Extends Path (from $I - 1$)**
   - Predecessor: $(t_1, t_2) = T[I-1][i_1][j], \quad \forall i_1 < i.$
   - Condition: $t_1 \leq ls(i)$ and $i \notin$ used_sets$[I-1][i_1][j]$.
   - Result:
   $$(\hat{t}_1, \hat{t}_2) = (\text{ret}(\max(t_1, es(i)), v, D[i]),\ t_2).$$

2. **Transition B: Drone 2 Extends Path (from $I-1$)**

   - Predecessor: $(t_1, t_2) = T[I-1][i][j_1], \quad \forall j_1 < j.$
   - Condition: $t_2 \le ls(j)$ and $j \notin$ used_sets$[I-1][i][j_1]$.
   - Result:
   $$(\hat{t}_1, \hat{t}_2) = (t_1,\ \mathrm{ret}(\max(t_2, es(j)), v, D[j]))\,.$$

3. **Transition C: Both Drones Extend Path (from $I-2$)**

   - Predecessor: $(t_1, t_2) = T[I-2][i_1][j_1], \quad \forall i_1 < i,\ j_1 < j \quad (I \ge 2).$
   - Condition: $t_1 \le ls(i)$, $t_2 \le ls(j)$, and $\{i, j\} \cap$ used_sets$[I-2][i_1][j_1] = \emptyset$.
   - Result:
   $$(\hat{t}_1, \hat{t}_2) = (\mathrm{ret}(\max(t_1, es(i)), v, D[i]),\ \mathrm{ret}(\max(t_2, es(j)), v, D[j]))\,.$$

The transition that minimizes the makespan determines the new DP entry and its predecessor:

$$T[I][i][j] = (\hat{t}_1^*, \hat{t}_2^*), \qquad \mathrm{prev}[I][i][j] = (i_1^*, j_1^*).$$

3.3 Experiment

This section empirically evaluates the performance of the proposed dynamic programming–based algorithms, highlighting their effectiveness in improving delivery coverage under different problem parameters.

3.3.1 Experimental Setup

Delivery Point Generation and Geometric Constraints
All experiments were conducted on proper instances of delivery points generated using a geometric instance-generation procedure based on the constraints described by Krizanc et al [4]. Multiple point sets were generated for each experiment to capture diverse geometric structures and ensure robust testing.

Common Configurations and Repetition
All experiments use the same truck and drone parameters, and computational environment described in the previous section. All experiments use 10 randomized, but identical point sets used for every algorithm trial to ensure fair pairwise comparisons. Performance was assessed based on *Number of Deliveries*, *Execution Time*, *Scalability*, *Sensitivity to Range and Velocity* and *Resource Efficiency* (minimum number of drones required to achieve full coverage). The following parameters were systematically varied:

Table 2: Simulation parameters for experiment 2

Parameter	**Values Tested**
Number of Delivery Points (N)	50, 60, **70**, 80, 90, 100
Drone Velocity (v)	1.2, 1.4, **1.6**, 1.8, 2.0, 2.2
Drone Range (R)	0.2, 0.4, **0.6**, 0.8, 1.0, 1.2
Number of Drones (K)	1, 2, 3, 4, 5, 6, 7, 8, 9, 10

The default configuration uses $N = 70$, $v = 1.6$, and $R = 0.6$, with k sweeping from 1 to 10.

3.3.2 Experimental Results

The four algorithms considered are Sequential 1DP, Parallel DP, Sequential 2DP and Sequential 2DP* with Sequential Greedy used as a benchmark.

(1) Number of Deliveries Comparison (Baseline Fleet Sweep)
For small to moderate fleets (1–8 drones), Sequential 2DP* consistently achieves the highest delivery counts. Sequential 1DP, Sequential 2DP, and Sequential Greedy perform slightly lower but remain competitive. For larger fleets (9–10 drones), Parallel DP surpasses Sequential 2DP*, demonstrating superior scaling. Sequential Greedy maintains reasonable coverage but is slightly below DP-based methods. The relative performance of all algorithms across different fleet sizes is illustrated in Figure 11.

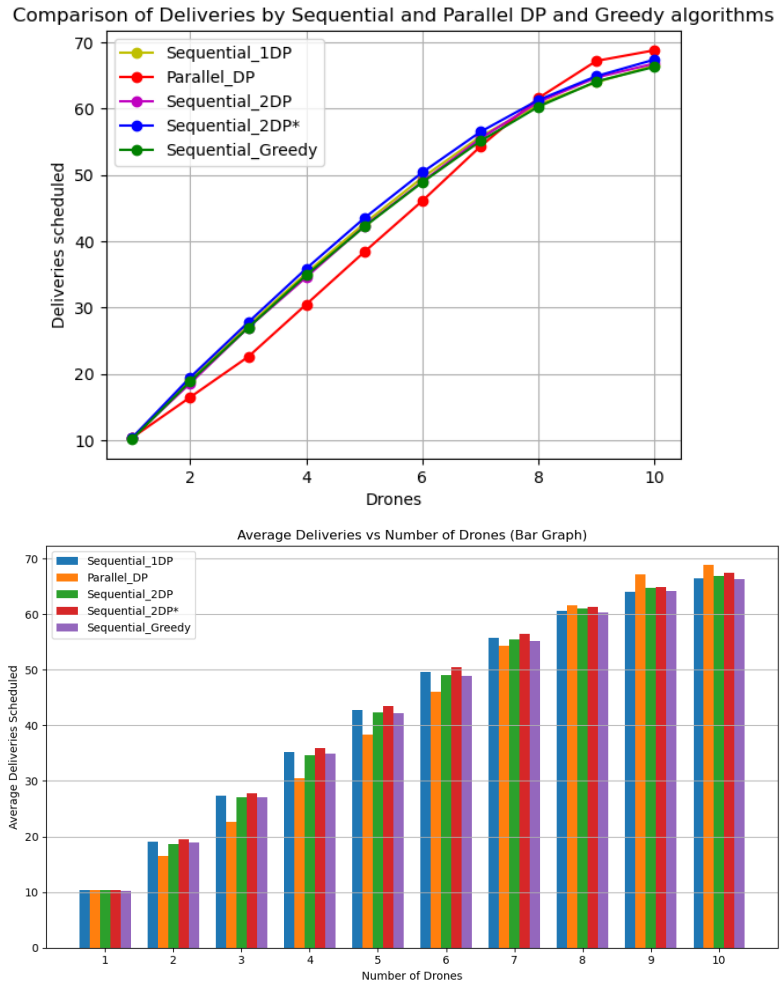


Figure 11: Comparison of the number of deliveries achieved by all algorithms across fleet sizes.

(2) Execution Time Comparison
The computational performance of the algorithms is categorized by complexity:

- High Cost (Sequential 2DP, 2DP*, $O(kN^5)$): Runtimes range from tens of seconds up to over 50 seconds per trial, making them prohibitively slow for large fleets or time-sensitive applications.

- Moderate Cost (Sequential 1DP, Parallel DP, $O(kN^3)$): These methods are practical for real-time scheduling, with execution times typically below 0.02 seconds even for large fleets.
- Low Cost (Sequential Greedy, $O(kN^2)$): Extremely fast, with runtimes in the order of milliseconds, suitable for real-time scheduling.

This highlights a direct trade-off: high-dimensional DP methods can achieve superior delivery coverage, especially for small fleets, but at the cost of prohibitive execution times. The execution time differences across all algorithms are summarized in Figure 12.

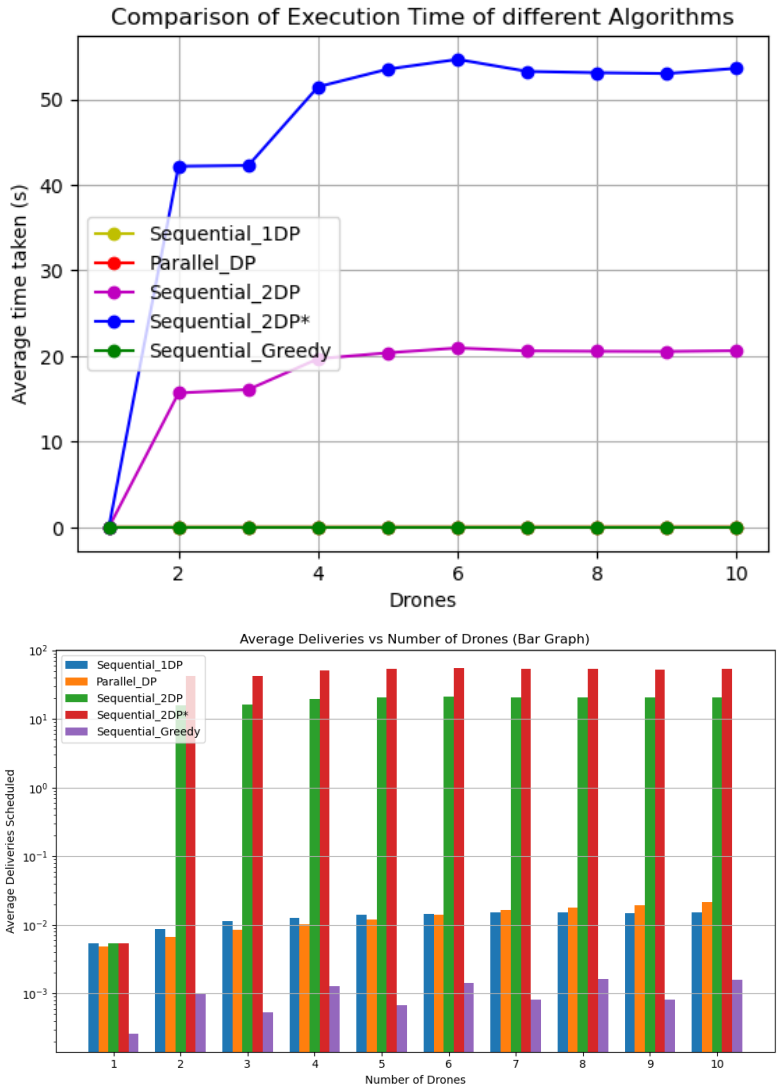


Figure 12: Execution time comparison of all algorithms across fleet sizes

(3) Number of Deliveries vs. Number of Points (N Sweep)
As N increases from 50 to 100: Sequential_2DP* maximizes deliveries across all point densities, maintaining its lead. Parallel_DP performance shows a relative decline as N increases. Sequential_1DP provides balanced performance, maintaining moderate coverage efficiently as the problem size scales. The comparative performance of the different algorithms as the number of points increases is shown in Figure 13.

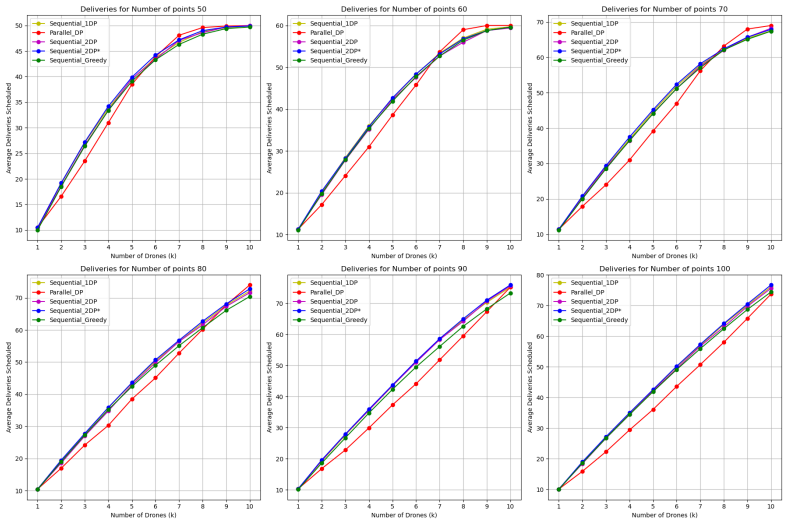


Figure 13: Comparison of delivery performance of different algorithms as the number of delivery points (N) increases

(4) Number of Deliveries vs. Drone Range (R Sweep)

Across all configurations, the number of completed deliveries increases with both drone range and fleet size, saturating once enough drones are available to cover all feasible points. For small ranges, all algorithms perform almost identically. For moderate ranges, Sequential-2DP* consistently achieves the highest delivery count, while Parallel-DP performs worse for small drone counts but improves as more drones are deployed. At larger ranges, the performance gap widens: Sequential-2DP* remains the strongest method, followed closely by Sequential-1DP and Sequential-2DP, whereas Parallel-DP significantly underperforms for 2–6 drones. Overall, Sequential-2DP* provides the most stable and superior performance across all ranges, while Parallel-DP struggles when the drone range is restrictive.

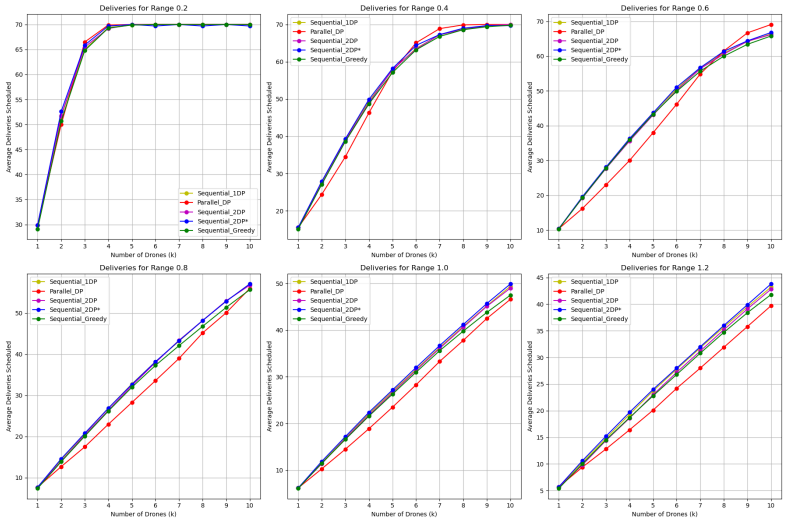


Figure 14: Comparison of delivery performance for different algorithms across varying drone ranges (R)

(5) Number of Deliveries vs. Drone Velocity (v Sweep)

For small fleets, Sequential_2DP* consistently achieves the highest average number of deliveries. As drone velocity increases, all algorithms see improved delivery counts. For larger fleets, Parallel_DP scales most effectively, taking advantage of increased speed to approach full delivery saturation, while sequential methods show diminishing returns. Across all velocities, Sequential_1DP provides steady but modest improvements and improves slightly over Sequential_2DP, and Sequential_Greedy remains competitive but generally underperforms compared to the optimized DP approaches. The impact of increasing drone velocity on the performance of different algorithms is shown in Figure 15.

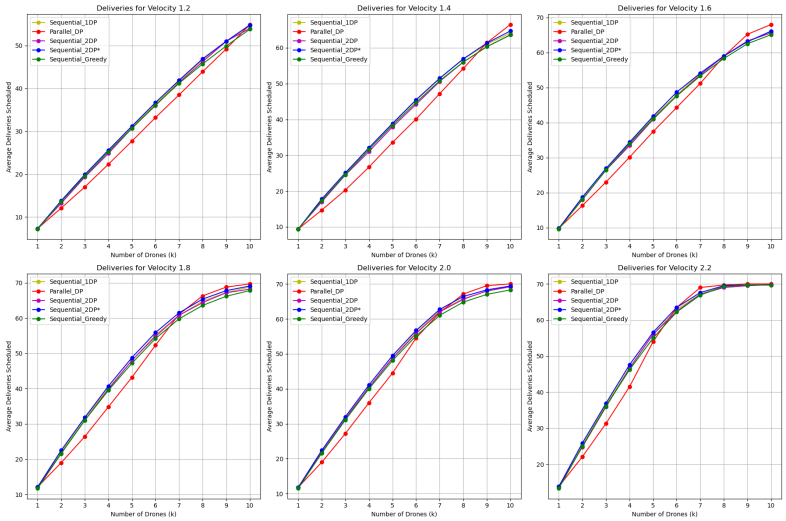


Figure 15: Comparison of delivery performance for different algorithms as drone velocity (v) increases

(6) Minimum-Drone Analysis (Resource Efficiency)

The resource-efficiency results reveal a clear and consistent trend across all tested point densities. The Parallel_DP algorithm requires the fewest drones in every scenario, making it the most resource-optimal method. Its advantage becomes more pronounced as the number of delivery points increases. The Sequential_1DP and Sequential_2DP algorithms consistently require larger drone counts, providing no efficiency gains over Parallel_DP. The enhanced dual-path heuristic Sequential_2DP* performs slightly better than the other DP heuristics but still falls short of matching the efficiency of Parallel_DP, especially at higher point densities. The Sequential_Greedy algorithm is generally the least efficient, often exhibiting the highest drone usage across the tested instances. Overall, the data clearly confirms that Parallel_DP is the most drone-efficient algorithm, while the Sequential DP variants and the greedy strategy tend to overuse drones as the number of delivery points increases. A comparative visualization of these trends is provided in Figure 16.

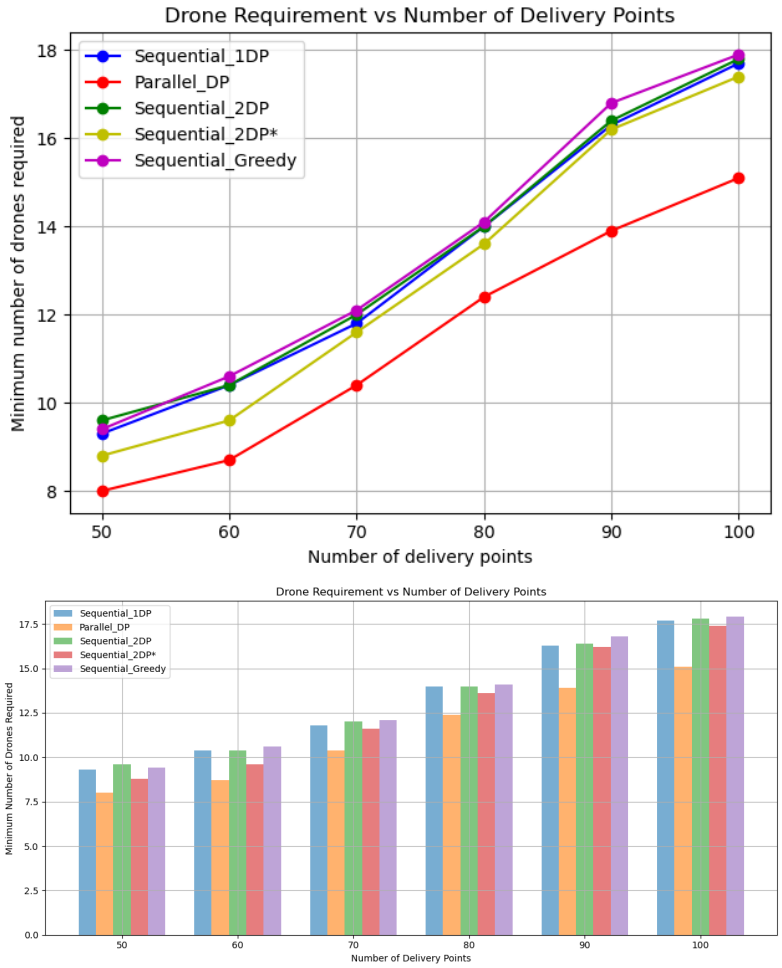


Figure 16: Minimum number of drones required by different algorithms to achieve full delivery coverage across varying point densities

Conclusion

- The Sequential_2DP* (3D DP) algorithm consistently achieves the highest number of completed deliveries for small to moderate fleet sizes (k) across most constraint settings (R, N, v), yielding the best solution quality among the evaluated methods in this regime. The Sequential_1DP algorithm performs slightly below Sequential_2DP*, while remaining competitive and significantly more efficient. In contrast, Parallel_DP is initially inefficient for small values of k but exhibits superior scalability as the fleet size increases, eventually matching or surpassing Sequential_2DP* for larger numbers of drones.

- A key practical distinction among the algorithms lies in their computational complexity. The dual-path 3D DP methods with $O(kN^5)$ time complexity quickly become operationally infeasible, with execution times reaching tens of seconds. In contrast, the $O(kN^3)$ algorithms (Sequential_1DP and Parallel_DP) consistently run in the millisecond range, making them suitable for larger problem instances.

- Parallel_DP requires the smallest number of drones across all tested point densities, making it the most resource-efficient approach. All sequential DP variants (Sequential_1DP, Sequential_2DP, and Sequential_2DP*) exhibit higher drone usage and become progressively less efficient as delivery density increases.

Overall, Sequential_1DP provides a strong compromise, achieving delivery counts close to those of higher-dimensional DP methods while maintaining significantly lower execution times.

Previous sections focused on coverage maximization under a moving-truck synchronization model. We now move to a different operational assumption: the truck may stop and wait for the drone. This changes what is meaningful to optimize: when full coverage is feasible, the central question becomes how to minimize the total completion time rather than how many points can be served. Next section develops algorithms for this time-minimization objective.

4 Truck-Drone Model: Truck can stop (Single drone)

Unlike previous sections, which maximize the number of completed deliveries, this chapter assumes a waiting synchronization model where the truck may stop and wait for the drone. Under this assumption, the objective shifts from coverage to time: we minimize the total completion time required to serve all deliveries. This shift also changes the algorithmic emphasis: we compare simple baselines, fast heuristics, and DP-style optimization to quantify the quality–runtime trade-off.

4.1 Problem Formulation

This section analyzes four distinct algorithms for optimizing a single-drone, single-truck delivery system under a waiting synchronization model, where the truck is allowed to stop and wait for the drone to return. The objective is to minimize the total completion time required to serve all points from a given set of random delivery points.

The four algorithms considered are: a naive baseline approach (Alg 1), a heuristic boundary-adjustment method (Alg 2), a fixed-order bottom-up dynamic programming algorithm (Alg 3), and an order-free exact search algorithm based on state-space exploration (Alg 4). All algorithms are evaluated on experimental instances consisting of randomly generated delivery points.

For each algorithm, we measure both the total delivery time and the computational execution time. These metrics are compared across varying numbers of delivery points, as well as different values of drone range and drone velocity, in order to assess the trade-offs between solution quality and computational efficiency.

Model assumptions. The truck moves at a constant velocity of 1 along the x-axis and may stop and wait for the drone to return. For each delivery $d_i = (x_i, y_i)$, the drone performs a single out-and-back trip, launched at position s_i and recovered at position r_i.

Under the waiting synchronization model considered here, the duration of the delivery segment is determined by the drone flight time

$$t_{\text{drone}} \;=\; \frac{\text{dist}((s_i,0),d_i) + \text{dist}(d_i,(r_i,0))}{v},$$

since the truck may wait at the recovery position if it arrives early. Feasibility of a synchronization interval (s_i, r_i) therefore requires that the drone can both complete its flight and rendezvous with the truck:

$$\text{dist}((s_i,0),d_i) + \text{dist}(d_i,(r_i,0)) \;\le\; R, \qquad t_{\text{drone}} \;\ge\; r_i - s_i.$$

The first condition enforces the drone's maximum flight range R, while the second guarantees that the drone remains airborne until the truck reaches the recovery position r_i.

4.2 Algorithms and Analysis

Having defined feasibility and the time-based objective, we now present algorithms that range from a naive baseline to DP and exact search, highlighting where additional optimization effort yields meaningful time reductions.

Common Inputs
The algorithms share a set of common inputs:

- **deliveries**: A list of delivery points in the form $[(x_1, y_1), (x_2, y_2), \dots]$.
- **R**: The maximum range the drone can travel on a single trip.
- **v**: The velocity of the drone (v_d).
- **max_truck_distance**: The maximum endpoint of the route, representing the farthest point the truck must reach.

4.2.1 Algorithm 1 (Naive Approach)

The Naive Algorithm operates under a simple approach. The truck moves to the x-coordinate of each delivery point and waits for the drone to perform a vertical round trip. The drone then moves vertically to the point and returns back to truck after serving the point. For each delivery, the drone travels a total distance of $2y_i$(y_i to the delivery point and y_i back to the truck). The time for the drone to complete this round trip is $2y_i/v_d$, where v_d is the drone's speed. The time taken for the truck to travel between two consecutive points ($i-1$ and i) is $x_i - x_{i-1}$ because velocity of the truck is 1. The total time for the truck's travel is the sum of these segments, which is equal to the maximum distance it needs to cover, which we can denote as max_truck_distance. The corresponding procedural steps for computing the total time under this Naive strategy are summarized in Algorithm 12.

Total Time: The total time for the algorithm is the sum of the truck's total travel time and the cumulative time of all the drone's round-trip times.

$$T_{total} = \text{max\_truck\_distance} + \sum_{i=1}^{n} \frac{2y_i}{v_d}.$$

Optimality: Not optimal. The truck waiting at each point is a significant source of inefficiency as it prevents the truck and drone from operating concurrently.

Algorithm 12: NAIVE ALGORITHM

Input: $deliveries$ (list of points (x, y)), drone speed v, truck path length $x_{\max}$
Output: $total\_Time$
$total\_Time \leftarrow x_{\max}$;
foreach (x, y) *in* $deliveries$ **do**
    $flight\_time \leftarrow \frac{2y}{v}$; `// Out-and-back drone trip`
    $total\_Time \leftarrow total\_Time + flight\_time$;
return $total\_Time$;

4.2.2 Algorithm 2 (Heuristic Boundary Adjustment)

Algorithm 2 is a heuristic optimization algorithm for the truck–drone delivery problem under a waiting synchronization model. The algorithm refines an initial schedule by iteratively adjusting the launch and return positions of the drone for each delivery in order to reduce the total completion time while preserving feasibility.

Unlike Algorithm 1, which initializes the schedule by fixing the launch and return positions to the delivery point's x-coordinate, Algorithm 2 allows these positions to expand outward along the truck's trajectory, subject to ordering and synchronization constraints.

Step 1: Initialization.

- Sort all delivery points $d_i = (x_i, y_i)$ by increasing x_i.
- For each delivery i, initialize the synchronization interval as $s_i = r_i = x_i$.
- Maintain a boolean array `boundary_reached` of size $2n$, where index $2i$ corresponds to the launch boundary s_i and index $2i + 1$ corresponds to the return boundary r_i. A boundary marked as reached can no longer be adjusted.
- Fix a step size $\Delta = 0.001$, which controls the granularity of boundary movement.

Step 2: Boundary selection. At each iteration, the algorithm scans all boundaries that are not yet locked and computes the decrement factor

$$df \;=\; \frac{x}{\sqrt{x^2 + y^2}},$$

where x is the horizontal distance between the current boundary position (s_i or r_i) and the delivery point x_i, and y is the vertical offset of the delivery.

The decrement factor represents the marginal increase in drone flight distance per unit horizontal movement of the truck. Boundaries with smaller df values incur a smaller increase in drone travel time and are therefore preferable to adjust. The boundary with the minimum df is selected for the next adjustment.

Step 3: Feasibility checks and boundary adjustment. Let the selected boundary correspond to delivery i. Determine the neighboring constraints:

$$r_{\text{prev}} = \begin{cases} 0, & i = 0, \\ r_{i-1}, & \text{otherwise}, \end{cases} \qquad s_{\text{next}} = \begin{cases} x_{\max}, & i = n-1, \\ s_{i+1}, & \text{otherwise}. \end{cases}$$

If the selected boundary is a launch boundary, attempt $s_i \leftarrow s_i - \Delta$. If the selected boundary is a return boundary, attempt $r_i \leftarrow r_i + \Delta$.

The attempted adjustment is rejected and the selected boundary is permanently locked if the proposed one-step update violates feasibility:

- *Ordering constraint (neighbor safety).* The update must satisfy $s_i - \Delta \geq r_{\text{prev}}$ for a launch update and $r_i + \Delta \leq s_{\text{next}}$ for a return update.
- *Drone range constraint.* Let (s_i', r_i') denote the candidate interval after applying the update. If

  $$\text{dist}((s_i', 0), d_i) + \text{dist}(d_i, (r_i', 0)) \;>\; R,$$

  then the update is infeasible.

- *Synchronization feasibility.* Let

$$t_1 \;=\; \frac{\mathrm{dist}((s_i',0),d_i)+\mathrm{dist}(d_i,(r_i',0))}{v}, \qquad t_2 \;=\; r_i'-s_i'.$$

  If $t_1 < t_2$, the drone cannot remain airborne long enough to rendezvous with the truck, and the update is infeasible.

If none of these conditions is violated, the update is accepted and the synchronization interval (s_i, r_i) is updated. The process repeats until all boundaries are locked.

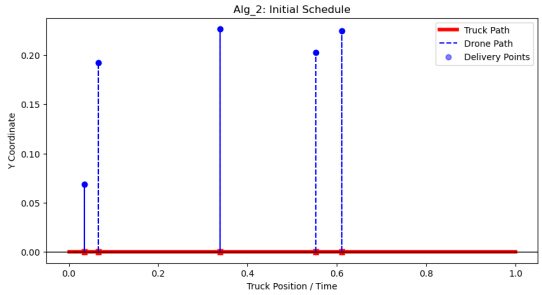


Figure 17: Initial schedule for the heuristic algorithm

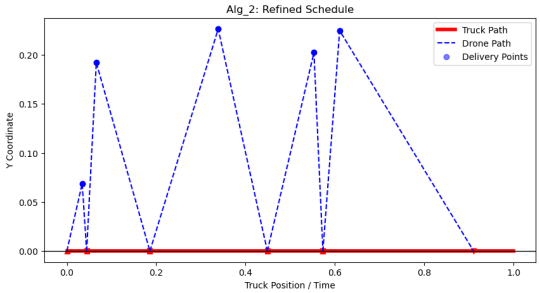


Figure 18: Final refined schedule

Step 4: Completion time computation. After the boundary adjustment phase terminates, the total completion time is computed by scanning deliveries in order:

- Add the truck travel since the previous return position, r_{prev}:

$$T \mathrel{+}= \; s_i - r_{\text{prev}}.$$

- Add the drone flight time:

$$T \mathrel{+}= \; \frac{\mathrm{dist}((s_i,0),d_i)+\mathrm{dist}(d_i,(r_i,0))}{v}.$$

After all deliveries are processed, if the truck has not yet reached $x_{\max}$, add $x_{\max} - r_{\text{prev}}$ to the total completion time.

Running time and optimality. As shown in Algorithm 13, the heuristic repeatedly selects the boundary with the minimum decrement factor and expands it by a fixed step size Δ. Each boundary moves monotonically and can be adjusted at most $O(x_{\max})$ distance, yielding

$$M \;=\; \frac{x_{\max}}{\Delta}$$

possible adjustment steps per boundary. Since each iteration scans all N deliveries, the total running time is

$$O(NM) \;=\; O\Big(N \cdot \frac{x_{\max}}{\Delta}\Big).$$

Algorithm 2 is a local optimization heuristic: it produces high-quality schedules in practice but does not guarantee global optimality.

Algorithm 13: Algorithm 2a: Initialization and Boundary Selection

Input: Deliveries $D = \{(x_i, y_i)\}$
Output: Initialized intervals SI and active boundary index min_idx
Sort D by increasing x-coordinate
$N \leftarrow |D|$
for $i \leftarrow 0$ **to** $N - 1$ **do**
  Initialize $SI[i] \leftarrow (d_i, x_i, x_i)$
end
Initialize boundary_reached$[1 \dots 2N] \leftarrow$ false
$min\_df \leftarrow \infty$, $min\_idx \leftarrow$ None
for $i \leftarrow 0$ **to** $N - 1$ **do**
  $(d, s, r) \leftarrow SI[i]$
  if *left boundary of* i *not reached* **then**
    compute $df = \frac{d_x - s}{\sqrt{(d_x - s)^2 + d_y^2}}$
    update (min_df, min_idx)
  end
  if *right boundary of* i *not reached* **then**
    compute $df = \frac{r - d_x}{\sqrt{(r - d_x)^2 + d_y^2}}$
    update (min_df, min_idx)
  end
end

4.2.3 Algorithm 3 (Dynamic Programming with Discretized Truck Positions)

Algorithm 3 is a dynamic programming algorithm for the truck–drone delivery problem under the same waiting synchronization model as Algorithms 1 and 2. Unlike the heuristic boundary-adjustment approach, Algorithm 3 enumerates all feasible synchronization choices (s'_j, r_j) on a discretized truck trajectory for a fixed order of deliveries. It returns an optimal schedule with respect to this fixed order and the chosen discretization granularity Δ, but it does not optimize over all permutations of the deliveries nor over continuous (non-discretized) launch/return positions.

The algorithm discretizes the truck's motion along the x-axis into uniform steps of size Δ and applies dynamic programming to determine the optimal launch and return positions for each delivery. For simplicity, we denote discretized truck positions directly by their coordinate values $s, r \in \{0, \Delta, 2\Delta, \dots, x_{\max}\}$. The pseudocode is provided in Algorithms 15 and 16.

Step 1: Discretization and preprocessing. Let

$$M \;=\; \left\lfloor \frac{x_{\max}}{\Delta} \right\rfloor + 1$$

denote the number of discretized truck positions. For each delivery j and each discretized launch position s, the algorithm precomputes the set of feasible return positions $r \geq s$ satisfying

$$\mathrm{dist}((s, 0), d_j) + \mathrm{dist}(d_j, (r, 0)) \;\leq\; R.$$

Algorithm 14: Algorithm 2b: Boundary Expansion and Schedule Evaluation

incr $\leftarrow 0.001$
$i \leftarrow \lfloor min\_idx/2 \rfloor$
$(d, s, r) \leftarrow SI[i]$
Determine neighbor bounds r_{prev} and s_{next}
if min_idx *even* **then**
    `// attempt to move launch boundary left`
    $s' \leftarrow s -$ incr
    $r' \leftarrow r$
    $d_{\text{tot}} \leftarrow \text{dist}((s', 0), d) + \text{dist}(d, (r', 0))$
    $t_1 \leftarrow d_{\text{tot}}/v$
    $t_2 \leftarrow r' - s'$
    if $s' < r_{prev}$ ***or*** $d_{tot} > R$ ***or*** $t_1 < t_2$ **then**
        mark boundary reached
    end
    else
        $s \leftarrow s'$
    end
else
    `// attempt to move return boundary right`
    $s' \leftarrow s$
    $r' \leftarrow r +$ incr
    $d_{\text{tot}} \leftarrow \text{dist}((s', 0), d) + \text{dist}(d, (r', 0))$
    $t_1 \leftarrow d_{\text{tot}}/v$
    $t_2 \leftarrow r' - s'$
    if $r' > s_{next}$ ***or*** $d_{tot} > R$ ***or*** $t_1 < t_2$ **then**
        mark boundary reached
    end
    else
        $r \leftarrow r'$
    end
end
Update $SI[i] \leftarrow (d, s, r)$
`// Schedule evaluation using the updated intervals`
$T \leftarrow 0$
$r_{prev} \leftarrow 0$
for $j \leftarrow 0$ **to** $|SI| - 1$ **do**
    $(d, s, r) \leftarrow SI[j]$
    $T \leftarrow T + (s - r_{prev})$
    $T \leftarrow T + \frac{\text{dist}((s,0),d)+\text{dist}(d,(r,0))}{v}$
    $r_{prev} \leftarrow r$
end
if $r_{prev} < x_{\max}$ **then**
    $T \leftarrow T + (x_{\max} - r_{prev})$
end
return SI, T

For each such pair (s, r), the corresponding drone flight time

$$t_{\text{drone}}(j, s, r) \;=\; \frac{\text{dist}((s,0), d_j) + \text{dist}(d_j, (r,0))}{v}$$

is stored for constant-time access during the dynamic programming phase.

Step 2: Dynamic programming formulation. The algorithm maintains a table

$$dp[j][s],$$

which represents the minimum remaining completion time required to serve deliveries $j, j+1, \ldots, N-1$, assuming the truck is currently at position s.

Two auxiliary tables, `choice_launch` and `choice_return`, are used to record the optimal launch and return positions for schedule reconstruction.

Base case. For the final delivery $j = N-1$, the algorithm considers all feasible launch positions $s' \geq s$ and feasible return positions $r \geq s'$. The total cost consists of:

- truck travel from s to s',
- the drone flight time for delivery j,
- truck travel from r to $x_{\max}$.

A candidate pair (s', r) is feasible only if

$$t_{\text{drone}}(j, s', r) \;\geq\; r - s',$$

which guarantees a valid rendezvous: the truck can reach the recovery position r no later than the drone returns. If the truck arrives earlier, it may wait at r; however, candidates where the drone would return before the truck reaches r are discarded.

Step 3: DP transition. For deliveries $j = N-2, N-3, \ldots, 0$, the algorithm computes

$$dp[j][s] \;=\; \min_{s' \geq s} \; \min_{r \in \mathcal{R}(j,s')} \left[s' - s + t_{\text{drone}}(j, s', r) + dp[j+1][r] \right],$$

subject to the same synchronization feasibility condition

$$t_{\text{drone}}(j, s', r) \;\geq\; r - s'.$$

Infeasible transitions are discarded. The positions (s', r) achieving the minimum value are stored for reconstruction.

Step 4: Schedule reconstruction. If $dp[0][0] < \infty$, the algorithm reconstructs an optimal schedule by starting at state $(j, s) = (0, 0)$ and repeatedly following the stored launch and return choices. The resulting schedule consists of launch–return pairs (s'_j, r_j) for each delivery.

If $dp[0][0] = \infty$, no feasible schedule exists under the given parameters.

Running time and optimality.

The time complexity of the optimal Dynamic Programming approach (Algorithm 3) is determined by the number of deliveries N and the number of discretized truck positions $M = \lfloor x_{\max}/\Delta \rfloor + 1$. The algorithm operates in two phases. First, the feasibility precomputation step, which calculates the drone travel time for all $O(M^2)$ launch-return pairs for each of the

N deliveries, incurs a cost of $\Theta(NM^2)$. Second, the DP table filling phase evaluates every possible launch position $s' \geq s$ and its associated set of feasible return indices, $F_j(s')$. This recurrence step is dominated by the triple summation $\sum_{j=0}^{N-1} \sum_{s=0}^{M-1} \sum_{s'=s}^{M-1} |F_j(s')|$. In the worst case, where every launch-return pair is feasible, the size of $|F_j(s')|$ is $\Theta(M - s')$, which yields a cost of $\Theta(NM^3)$. Therefore, the total time complexity of the algorithm is dominated by the DP recurrence, resulting in a worst-case bound of $T = \Theta(NM^3)$.

Unlike Algorithm 2, Algorithm 3 is exact for a fixed delivery order under the discretized model. Specifically, for the given ordered list of deliveries $(d_0, d_1, \ldots, d_{N-1})$, it computes a minimum-time schedule over all discretized launch/return choices (s'_j, r_j) that respect this order. Therefore, Algorithm 3 guarantees global optimality with respect to the chosen order and discretization granularity Δ, but it does not optimize over all possible permutations of the deliveries.

Algorithm 15: Alg 3: Feasibility Precomputation

Input: Deliveries $D = \{(x_j, y_j)\}_{j=0}^{N-1}$, Drone range R, Drone speed v, Truck horizon $x_{\max}$, Discretization step Δ

Output: Feasible return sets $F_j(s)$ and drone travel times $\tau_j(s, r)$

$M \leftarrow \left\lfloor \frac{x_{\max}}{\Delta} \right\rfloor + 1$

for $j \leftarrow 0$ **to** $N - 1$ **do**

  for $s \leftarrow 0$ **to** $M - 1$ **do**

    $F_j(s) \leftarrow \emptyset$

    for $r \leftarrow s$ **to** $M - 1$ **do**

      $x_s \leftarrow s \cdot \Delta$

      $x_r \leftarrow r \cdot \Delta$

      $d_1 \leftarrow \sqrt{(x_s - x_j)^2 + y_j^2}$

      $d_2 \leftarrow \sqrt{(x_r - x_j)^2 + y_j^2}$

      if $d_1 + d_2 \leq R$ **then**

        $F_j(s) \leftarrow F_j(s) \cup \{r\}$

        $\tau_j(s, r) \leftarrow \dfrac{d_1 + d_2}{v}$

return $\{F_j(s)\}, \{\tau_j(s, r)\}$

Algorithm 16: Alg 3: Dynamic Programming and Reconstruction

Input: Feasible return sets $F_j(s)$, Drone times $\tau_j(s, r)$, $x_{\max}$, Δ
Output: Optimal schedule and completion time
Initialize DP table $dp[j][s] \leftarrow \infty$
Initialize decision tables launch$[j][s]$, return$[j][s] \leftarrow -1$
Base Case (last delivery $j = N - 1$)
for $s \leftarrow 0$ **to** $M - 1$ **do**
  for $s' \leftarrow s$ **to** $M - 1$ **do**
    drive $\leftarrow (s' - s)\Delta$
    foreach $r \in F_{N-1}(s')$ **do**
      if $\tau_{N-1}(s', r) \geq (r - s')\Delta$ **then**
        $T \leftarrow$ drive $+ \tau_{N-1}(s', r) + (x_{\max} - r\Delta)$
        if $T < dp[N-1][s]$ **then**
          $dp[N-1][s] \leftarrow T$
          launch$[N-1][s] \leftarrow s'$
          return$[N-1][s] \leftarrow r$
Recursive Case
for $j \leftarrow N - 2$ **downto** 0 **do**
  for $s \leftarrow 0$ **to** $M - 1$ **do**
    for $s' \leftarrow s$ **to** $M - 1$ **do**
      drive $\leftarrow (s' - s)\Delta$
      foreach $r \in F_j(s')$ **do**
        if $\tau_j(s', r) \geq (r - s')\Delta$ **then**
          $T \leftarrow$ drive $+ \tau_j(s', r) + dp[j+1][r]$
          if $T < dp[j][s]$ **then**
            $dp[j][s] \leftarrow T$
            launch$[j][s] \leftarrow s'$
            return$[j][s] \leftarrow r$
Reconstruction
$s \leftarrow 0$
for $j \leftarrow 0$ **to** $N - 1$ **do**
  $s' \leftarrow$ launch$[j][s]$
  $r \leftarrow$ return$[j][s]$
  Append $(s'\Delta, r\Delta)$ to schedule
  $s \leftarrow r$
return *schedule,* $dp[0][0]$

4.2.4 Algorithm 4 (Order-Free Exact Search via State-Space Dijkstra)

Algorithm 4 extends Algorithm 3 by removing the restriction of a fixed delivery order. It formulates the truck–drone delivery problem as a shortest-path search over a discrete state space and applies Dijkstra's algorithm to compute an optimal schedule under the waiting synchronization model.

Unlike Algorithm 3, which optimizes launch and return positions for a given delivery or-

der, Algorithm 4 explicitly explores all permutations of deliveries together with all feasible discretized launch and return positions, and therefore optimizes both the order of deliveries and the synchronization choices.

Step 1: Discretization and preprocessing. The truck's trajectory is discretized into

$$M \;=\; \left\lfloor \frac{x_{\max}}{\Delta} \right\rfloor + 1$$

positions. For simplicity, discretized truck positions are denoted directly by their coordinate values $s, r \in \{0, \Delta, 2\Delta, \ldots, x_{\max}\}$. For each delivery j and each discretized launch position s, the algorithm precomputes the set of feasible return positions

$$F_j(s) = \{\, r \geq s \mid \mathrm{dist}((s,0), d_j) + \mathrm{dist}(d_j, (r,0)) \leq R \,\},$$

together with the corresponding drone flight times

$$t_{\text{drone}}(j, s, r) = \frac{\mathrm{dist}((s,0), d_j) + \mathrm{dist}(d_j, (r,0))}{v}.$$

Step 2: State-space formulation. Algorithm 4 represents partial schedules using states of the form

$$(\text{mask},\ s),$$

where:

- mask is a bitmask indicating which deliveries have already been completed, and
- s is the discretized truck position at the end of the last completed delivery.

The initial state is $(0, 0)$, corresponding to no deliveries served and the truck at the origin. A terminal state is any (full, s), where all deliveries have been served.

Step 3: Transitions and costs. From a state (mask, s), the algorithm considers any unserved delivery j and any launch position $s' \geq s$. For each feasible return position $r \in F_j(s')$, a transition is added provided that the synchronization condition enforced in the pseudocode holds:

$$t_{\text{drone}}(j, s', r) \;\geq\; r - s'.$$

The cost of this transition is

$$s' - s \;+\; \max(t_{\text{drone}}(j, s', r), r - s')\,,$$

which accounts for the truck's travel to the launch position and the duration of the synchronized delivery segment. The resulting successor state is

$$(\text{mask} \cup \{j\},\ r).$$

Step 4: Shortest-path search. All states and transitions define a directed weighted graph. Algorithm 4 applies Dijkstra's algorithm over this state space, maintaining the best-known completion time for each reachable state. Once all deliveries are served, the algorithm adds the final truck travel from the current position to $x_{\max}$ and selects the terminal state with minimum total time.

Step 5: Schedule reconstruction. Using parent pointers stored during the search, the algorithm reconstructs the delivery order and the corresponding launch–return pairs (s'_j, r_j). Unlike Algorithm 3, the resulting schedule may serve deliveries in any order.

Worst-case runtime of Algorithm 4. Algorithm 4 performs a shortest-path search over states of the form (mask, s), where mask encodes the set of completed deliveries and s is a discretized truck position. The state space contains $2^N \cdot M$ states.

From each state, up to $O(NM^2)$ transitions may be explored, corresponding to choices of the next delivery, launch position, and return position. Using Dijkstra's algorithm with a binary heap, the resulting worst-case runtime is

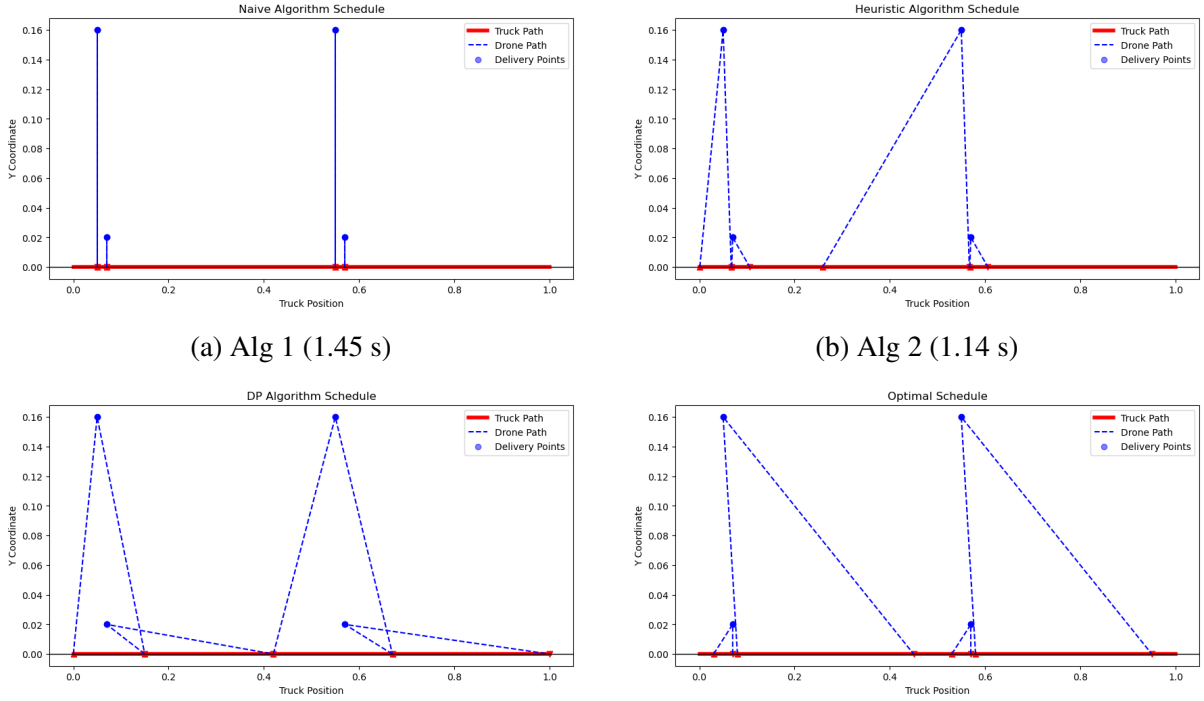
$$O\big(N\,2^N\,M^3 \log(2^N M)\big)\,,$$

which dominates the $\Theta(NM^2)$ preprocessing cost.

Algorithm 17 presents the pseudocode for the proposed exact discretized truck–drone scheduling algorithm, which computes an optimal delivery schedule using Dijkstra's shortest-path framework.

Due to the exponential growth of the state space, Algorithm 4 is computationally feasible only for instances with a small number of delivery points and is therefore used solely as a benchmark for evaluating heuristic approaches.

The schedules shown in Figure 19 correspond to the problem instance with drone range $R = 0.6$, drone speed $v = 1.6$, and delivery locations $D = \{(0.05, 0.16), (0.07, 0.02), (0.55, 0.16), (0.57, 0.02)\}$. Notice that the schedule produced by Algorithm 4 does not follow the order of the x-coordinates of the delivery locations.



(a) Alg 1 (1.45 s)

(b) Alg 2 (1.14 s)

(c) Alg 3 (1.08 s)

(d) Alg 4 (1.00 s)

Figure 19: Comparison of delivery schedules produced by Alg 1–Alg 4 for the same problem instance. Alg 4 achieves the minimum completion time and serves as the optimal benchmark.

Algorithm 17: Alg 4: Exact Discretized Truck–Drone Scheduling via Dijkstra

Input: Delivery points $D = \{(x_j, y_j)\}_{j=1}^{N}$, drone range R, drone speed v, maximum truck position $x_{\max}$, discretization step Δx, truck speed v_T
Output: Optimal delivery schedule and minimum completion time
Function `IdxToX(`i`)`**:**
  return $i \cdot \Delta x$
$M \leftarrow \lfloor x_{\max}/\Delta x \rfloor + 1$;
INF $\leftarrow +\infty$;

Precomputation of feasible drone returns;
for $j \leftarrow 1$ **to** N **do**
  for $s \leftarrow 0$ **to** $M-1$ **do**
    for $r \leftarrow s$ **to** $M-1$ **do**
      $d_1 \leftarrow \|(\texttt{IdxToX}(s), 0) - (x_j, y_j)\|$;
      $d_2 \leftarrow \|(\texttt{IdxToX}(r), 0) - (x_j, y_j)\|$;
      if $d_1 + d_2 \le R$ **then**
        store r in `feasible_returns`$[j][s]$;
        `drone_time`$[j][s][r] \leftarrow (d_1 + d_2)/v$;

Dijkstra on state space (**`mask`**, **`truck_idx`**);
`full` $\leftarrow 2^N - 1$;
Initialize priority queue PQ;
`dist`$[(0,0)] \leftarrow 0$;
Push $(0, 0, 0)$ into PQ;
while PQ *not empty* **do**
  $(t, \texttt{mask}, i) \leftarrow$ Extract-Min(PQ);
  if $t \neq$ `dist`$[(mask, i)]$ **then**
    continue
  if `mask` $=$ `full` **then**
    break
  foreach *unserved delivery* j **do**
    for $s \leftarrow i$ **to** $M-1$ **do**
      $t_{\text{drive}} \leftarrow (\texttt{IdxToX}(s) - \texttt{IdxToX}(i))/v_T$;
      foreach $r \in$ `feasible_returns`$[j][s]$ **do**
        $t_D \leftarrow$ `drone_time`$[j][s][r]$;
        $t_T \leftarrow (\texttt{IdxToX}(r) - \texttt{IdxToX}(s))/v_T$;
        if $t_D < t_T$ **then**
          continue
        $t' \leftarrow t + t_{\text{drive}} + \max(t_D, t_T)$;
        $\texttt{mask}' \leftarrow \texttt{mask} \cup \{j\}$;
        if $t' <$ `dist`$[(mask', r)]$ **then**
          update `dist` and `parent`;
          Push $(t', \texttt{mask}', r)$ into PQ;

Termination and reconstruction;
Select $(\texttt{full}, i)$ minimizing $\texttt{dist}[(\texttt{full}, i)] + (x_{\max} - \texttt{IdxToX}(i))/v_T$;
Reconstruct schedule using `parent` pointers;
return schedule and optimal completion time;

4.3 Experiment

We next evaluate these algorithms using common metrics: total delivery time (solution quality) and execution time (computational cost), across varying instance sizes and parameter settings.

4.3.1 Experimental Setup

To evaluate the performance of the first three algorithms, we conducted a series of computational experiments with the following setup:

- **Delivery Points, Truck and Drone Parameters and Hardware:** All delivery-generation procedures, truck/drone parameters, and hardware settings follow Experiment 2.3. The only difference is the delivery-point generation range in the y-direction: in Experiment 3 we enforce $0 \leq y \leq R/2$ (whereas Experiments 1–2 use the original y-range described in Experiment 2.3).
- **Evaluation Metrics:**
  1. *Total Delivery Time*: Time taken to complete all deliveries for a given scenario.
  2. *Execution Time*: CPU time required to compute the schedule for each algorithm.
- **Experimental Variations:** The following parameters were systematically varied:

Table 3: Simulation parameters for experiment 3

Parameter	**Values Tested**
Number of Delivery Points (N)	20, 40, 60, 80, **100**, 120, 140, 160, 180, 200
Drone Velocity (v)	1.2, 1.4, **1.6**, 1.8, 2.0, 2.2, 2.4, 2.6, 2.8, 3.0
Drone Range (R)	0.2, 0.4, **0.6**, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0

  The default configuration uses $N = 100$, $v = 1.6$, and $R = 0.6$.

- **Repetition and Averaging:**
  - For each experimental configuration, 10 random instances were generated, and results were averaged to reduce stochastic variability and ensure statistically meaningful conclusions.
- **Experimental Workflow:**
  1. A set of delivery points was generated randomly within the defined x and y ranges.
  2. All three algorithms were executed on the same set of points to obtain a delivery schedule.
  3. Total delivery time and execution time were recorded for each algorithm.
  4. The process was repeated for multiple trials, and the results were averaged.
  5. Performance metrics were plotted against experimental variables (number of points, drone velocity, and drone range) to analyze algorithm efficiency, scalability, and robustness.

4.3.2 Experimental Results

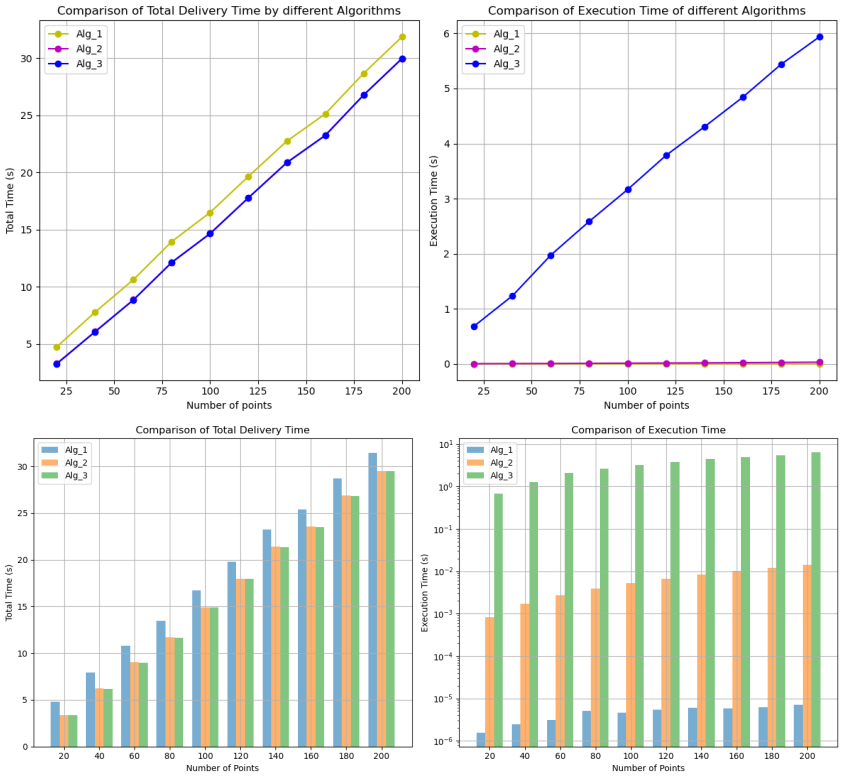


Figure 20: Total Delivery Time and Execution Time vs. Number of Points

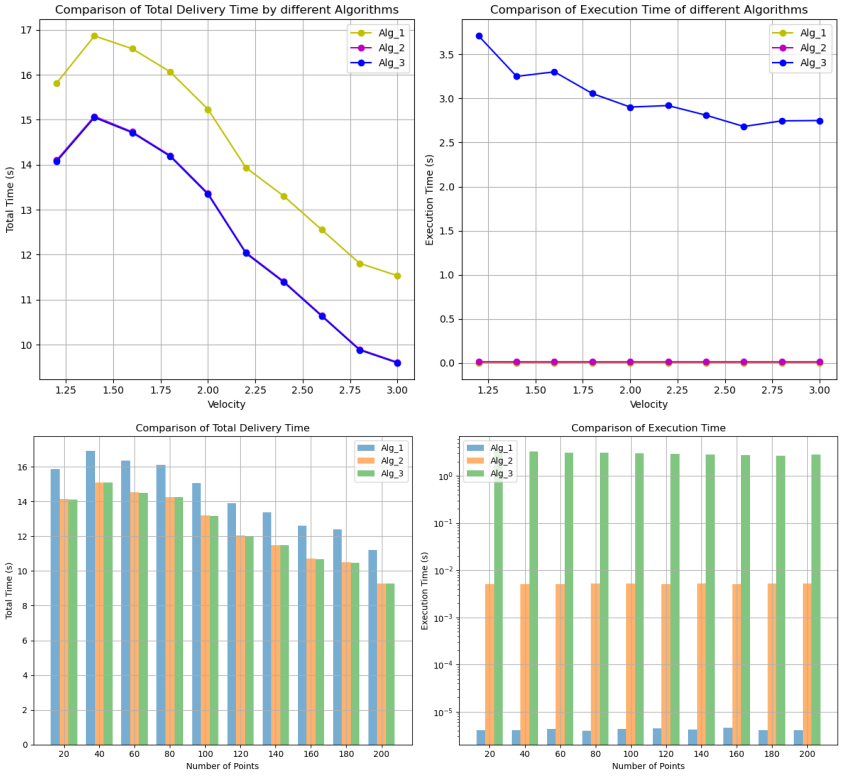


Figure 21: Total Delivery Time and Execution Time vs. Velocity

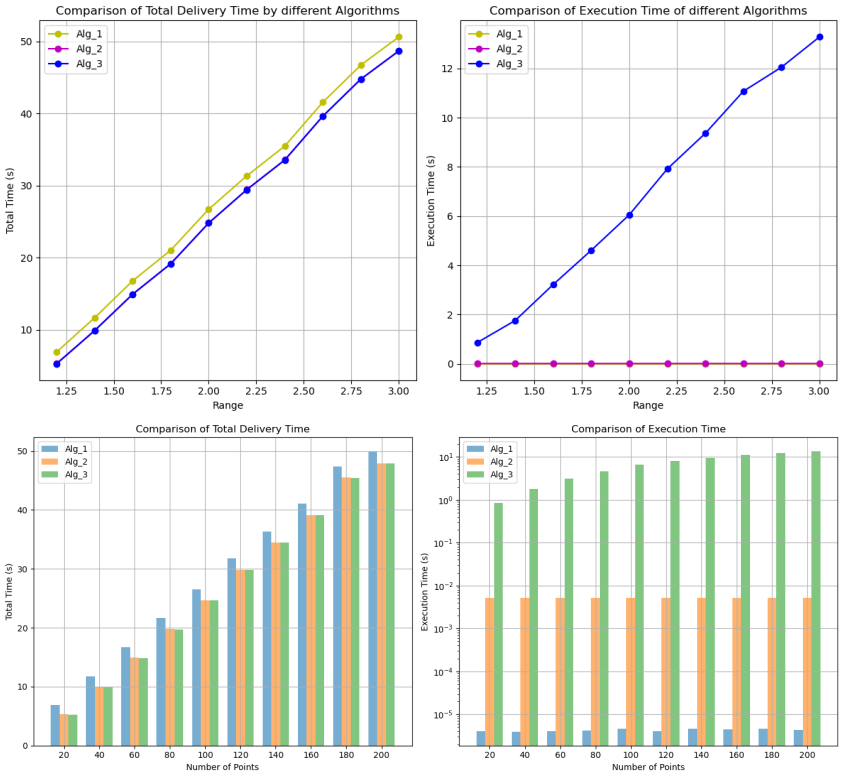


Figure 22: Total Delivery Time and Execution Time vs. Range

Total Delivery Time

Based on Figures 20, 21, and 22, the algorithms perform as follows with respect to total delivery time:

1. **Dynamic Programming (Alg 3)**: Consistently achieves the lowest total delivery time among the tested methods. This is expected, since Alg 3 optimizes over all feasible discretized launch and return positions for a *fixed delivery order*, yielding an optimal solution within this restricted model.

2. **Heuristic (Alg 2)**: Produces near-optimal results that are very close to those of Alg 3, with only minor increases in total time in some instances. Its boundary-adjustment strategy effectively balances truck motion and drone flight time, but it does not guarantee optimality.

3. **Naive (Alg 1)**: Performs significantly worse than Alg 2 and Alg 3, as it does not exploit synchronization between the truck and drone and fixes launch and return positions at the delivery points.

These results demonstrate that incorporating synchronization and optimization of launch and return positions (Alg 2 and Alg 3) substantially reduces total delivery time compared to the naive approach.

Computational Execution Time

Execution times illustrate the trade-off between solution quality and computational cost, as shown in Figures 20, 21, and 22:

1. **Naive (Alg 1)**: Fastest by far, with execution times in the microsecond range, due to its simple, non-iterative computation.

2. **Heuristic (Alg 2)**: Slower than Alg 1, with execution times in the millisecond range. Its iterative refinement incurs additional overhead, but remains efficient even for large delivery sets.

3. **Dynamic Programming (Alg 3)**: Slowest among the three, with execution times in the seconds range. The dynamic programming procedure evaluates a large number of discretized synchronization choices, resulting in substantially higher computational cost.

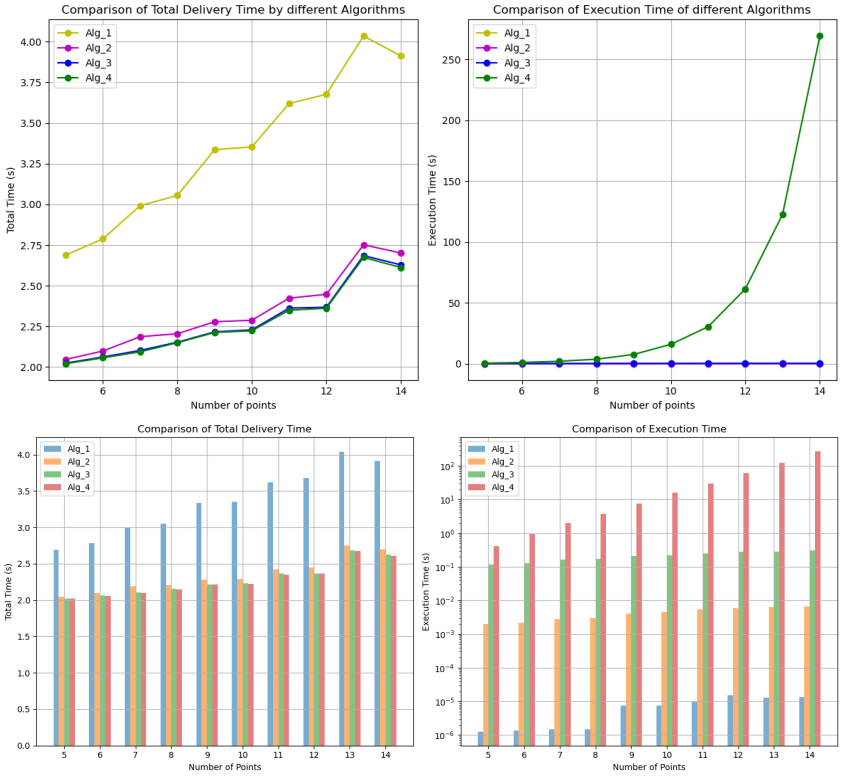


Figure 23: Comparison of all algorithms including Alg 4

As shown in Figure 23, even for small instances, Alg 4 does not provide a significant improvement in total delivery time compared to Alg 3, while incurring an exponentially larger execution time as the number of delivery points increases.

Conclusion

In summary, Alg 3 produces the best solutions for the fixed-order, discretized model but at a significant computational cost. Alg 2 offers an attractive trade-off, achieving solutions close to Alg 3 while running much faster, making it suitable for large-scale instances where execution time is critical. Alg 1 is computationally trivial but yields substantially inferior delivery times. Alg 4 serves primarily as a benchmark for small instances, validating the quality of the solutions obtained by the other algorithms.

5 Conclusion

This study presented a comprehensive analysis of algorithms for truck–drone delivery scheduling under multiple problem settings and modeling assumptions. The investigation progressed

through three main phases, each addressing a distinct level of algorithmic complexity and practical relevance.

First, the single-drone model was extended to multi-drone fleets with k drones, with the objective of maximizing the number of completed deliveries. For randomly located delivery points, two greedy heuristics were developed: the Sequential Greedy Algorithm and the Parallel Greedy Algorithm. The Sequential Greedy approach was shown to achieve a minimum approximation factor of $1 - \left(1 - \frac{1}{2k}\right)^k$, which converges to $1 - e^{-1/2} \approx 39.35\%$ as $k \to \infty$. Empirical results further demonstrated that the sequential strategy consistently outperforms the parallel approach in solution quality, execution time, and overallresource utilization.

Next, the study focused on geometrically constrained *proper instances*, which permit more structured Dynamic Programming (DP) techniques based on the classical $O(n^3)$ single-drone solution. Four algorithms were analyzed: Sequential 1-DP, Parallel DP, Sequential 2-DP, and Sequential 2-DP*. Sequential 1-DP achieved a tighter approximation factor of $1 - \left(1 - \frac{1}{k}\right)^k$, approaching $1 - e^{-1} \approx 63.21\%$ as $k \to \infty$. While the higher-dimensional DP variants (notably Sequential 2-DP*) produced the best solution quality for small fleets, their computational cost grows rapidly. In contrast, the $O(kN^3)$ Parallel DP algorithm offered a more scalable alternative for larger fleets, illustrating the trade-off between marginal gains in solution quality and computational efficiency. Overall, Sequential_1DP appears to balance solution quality and runtime, approaching the performance of more elaborate DP schemes without incurring their computational overhead.

Finally, the study examined the problem of minimizing total delivery time for a single drone under a waiting synchronization model, where the truck is allowed to stop and wait for the drone. Four algorithms were considered: a naive baseline (Alg 1), a heuristic boundary adjustment method (Alg 2), a fixed-order dynamic programming approach (Alg 3), and an order-free exact search method (Alg 4). Alg 3 optimizes the discretized launch and return positions for a given delivery order and consistently achieves the lowest total delivery time among fixed-order methods. Alg 2 produces near-optimal solutions at a fraction of the computational cost, making it well suited for large-scale or time-sensitive applications. Alg 4 removes the fixed-order restriction and performs an exhaustive search over all delivery permutations, serving as a validation benchmark for small instances but becoming computationally infeasible as the number of deliveries increases. Alg 1, while computationally trivial, yields substantially inferior schedules.

Overall, the results reveal a clear hierarchy among algorithmic approaches. Exact dynamic programming and state-space search methods provide strong optimality guarantees but incur high computational costs. Heuristic methods strike an effective balance by delivering high-quality solutions with excellent scalability, while naive strategies offer speed at the expense of solution quality. These findings provide practical guidance for selecting appropriate truck–drone scheduling algorithms based on problem size, performance requirements, and available computational resources.

6 Future Work and Limitations

The research presented here, while providing rigorous solutions for scheduling collaborative truck-multi-drone delivery, is subject to several key limitations and suggests clear paths for future work. The current models rely on simplified assumptions, including a fixed straight-line truck path, constant velocities, and the assumption of instantaneous drone operations (loading, recharging, and unloading). These simplifications restrict direct applicability to real-world

logistics involving dynamic road networks and variable conditions. Furthermore, the most powerful optimization techniques, such as the $O(kN^5)$ Dynamic Programming methods developed for proper instances, suffer from high computational complexity, making them impractical for large-scale operations and affirming the necessity of favoring efficient heuristics like the Sequential Greedy approach. The instance in which the truck is allowed to stop for drone operations can be naturally extended to accommodate multiple drones, allowing exploration of better collaborative strategies and improved delivery efficiency. Future work should address these limitations by integrating multi-visit drone capabilities, considering real-world dynamics like time-varying speeds and non-negligible operating times, and moving toward joint optimization of both the truck's path and the drone schedules.

7 Appendix

Code and results for all the algorithms and experiments presented in this thesis can be accessed at the following link: https://github.com/JerryMathew07/Thesis.

References

[1] F. Betti Sorbelli, F. Coró, S. K. Jana, P. G. Konda, V. G. S. Polapragada, and B. Sahoo, "Greedy Algorithms for Scheduling Package Delivery with Multiple Drones," in *Proceedings of the International Conference on Distributed Computing and Networking (ICDCN)*, 2022.

[2] T. Benarbia and K. Kyamakya, "A Literature Review of Drone-Based Package Delivery Logistics Systems and Their Implementation Feasibility," *Sustainability*, vol. 14, no. 1, p. 360, 2022, doi:10.3390/su14010360.

[3] Q. M. Ha, D. M. Vu, X. T. Le, and M. H. Ha, "The traveling salesman problem with multi-visit drone," *Journal of Computer Science and Cybernetics*, vol. 37, no. 4, pp. 465–493, 2021, doi:10.15625/1813-9663/15863.

[4] D. Krizanc, L. Narayanan, J. Opatrny, and D. Pankratov, "The En Route Truck-Drone Delivery Problem," *Theoretical Computer Science*, vol. 1006, art. no. 113943, 2024.

[5] C. C. Murray and M. Chu, "The flying sidekick traveling salesman problem," *Transportation Research Part C: Emerging Technologies*, vol. 54, pp. 86–104, 2015, doi:10.1016/j.trc.2015.03.005.

[6] C. C. Murray and R. Raj, "The multiple flying sidekicks traveling salesman problem: Parcel delivery with multiple drones," *Transportation Research Part C: Emerging Technologies*, vol. 110, pp. 368–398, 2020, doi:10.1016/j.trc.2019.11.003.

[7] V. Garg, S. Niranjan, V. Prybutok, T. Pohlen, and D. Gligor, "Drones in last-mile delivery: A systematic review on efficiency, accessibility, and sustainability," *Transportation Research Part D: Transport and Environment*, vol. 123, p. 103831, 2023, doi:10.1016/j.trd.2023.103831.

[8] A. Seidakhmetov and O. F. Valilai, "Drone based delivery system: Restrictions and limitations," in *Proceedings of the Hamburg International Conference of Logistics (HICL)*, 2022, pp. 351–373.

[9] J. K. Stolaroff, C. Samaras, E. R. O'Neill, and A. Lubers, "Energy use and greenhouse gas emissions of drones for commercial package delivery," *Nature Communications*, vol. 9, no. 1, p. 409, 2018, doi:10.1038/s41467-017-02411-5.

[10] J. Zhang, *Economic and environmental impacts of drone delivery*. Ph.D. dissertation, University of Missouri–St. Louis, 2021.

[11] H. Li, J. Chen, F. Wang, and Y. Zhao, "Truck and drone routing problem with synchronization on arcs," *Naval Research Logistics (NRL)*, vol. 69, no. 6, pp. 884–901, 2022, doi:10.1002/nav.22053.

[12] T. Thomas, S. Srinivas, and C. Rajendran, "Collaborative truck multi-drone delivery system considering drone scheduling and en route operations," *Annals of Operations Research*, vol. 339, pp. 693–739, 2024, doi:10.1007/s10479-023-05418-y.

[13] M. Marinelli, L. Caggiani, M. Ottomanelli, and M. Dell'Orco, "En route truck–drone parcel delivery for optimal vehicle routing strategies," *IET Intelligent Transport Systems*, vol. 12, no. 4, pp. 253–261, 2018, doi:10.1049/iet-its.2017.0227.

[14] A. Otto, N. Agatz, J. F. Campbell, B. L. Golden, and E. Pesch, "Optimization approaches for civil applications of unmanned aerial vehicles (UAVs) or aerial drones: A survey," *Networks*, vol. 72, no. 4, pp. 411–458, 2018, doi:10.1002/net.21818.

[15] S. H. Chung, B. Sah, and J. Lee, "Optimization for drone and drone-truck combined operations: A review of the state of the art and future directions," *Computers & Operations Research*, vol. 123, p. 105004, 2020, doi:10.1016/j.cor.2020.105004.

[16] G. Macrina, L. Di Puglia Pugliese, F. Guerriero, and G. Laporte, "Drone-aided routing: A literature review," *Transportation Research Part C: Emerging Technologies*, vol. 120, p. 102762, 2020, doi:10.1016/j.trc.2020.102762.

[17] S. Dang, Y. Liu, Z. Luo, Z. Liu, and J. Shi, "A Survey of the Routing Problem for Cooperated Trucks and Drones," *Drones*, vol. 8, no. 10, p. 550, 2024, doi:10.3390/drones8100550.

[18] N. Agatz, P. Bouman, and M. Schmidt, "Optimization Approaches for the Traveling Salesman Problem with Drone," *Transportation Science*, vol. 52, no. 4, pp. 965–981, 2018, doi:10.1287/trsc.2017.0791.

[19] J. G. Carlsson and S. Song, "Coordinated Logistics with a Truck and a Drone," *Management Science*, vol. 64, no. 9, pp. 4052–4069, 2018, doi:10.1287/mnsc.2017.2824.

[20] N. Boysen, D. Briskorn, S. Fedtke, and S. Schwerdfeger, "Drone delivery from trucks: Drone scheduling for given truck routes," *Networks*, vol. 72, no. 4, pp. 506–527, 2018, doi:10.1002/net.21847.

[21] S. Tamke and U. Buscher, "A branch-and-cut algorithm for the vehicle routing problem with drones," *Transportation Research Part B: Methodological*, vol. 144, pp. 174–203, 2021, doi:10.1016/j.trb.2020.11.011.

[22] Z. Wang and J.-B. Sheu, "Vehicle routing problem with drones," *Transportation Research Part B: Methodological*, vol. 122, pp. 350–364, 2019, doi:10.1016/j.trb.2019.03.005.

[23] H. Zhou, H. Qin, C. Cheng, and L.-M. Rousseau, "An exact algorithm for the two-echelon vehicle routing problem with drones," *Transportation Research Part B: Methodological*, vol. 168, pp. 124–150, 2023, doi:10.1016/j.trb.2023.01.002.

[24] S. Poikonen, B. Golden, and E. A. Wasil, "A branch-and-bound approach to the traveling salesman problem with a drone," *INFORMS Journal on Computing*, vol. 31, no. 2, pp. 335–346, 2019, doi:10.1287/ijoc.2018.0826.

[25] A. Vásquez, G. Angulo, and M. A. Klapp, "An exact method for the traveling salesman problem with a drone based on decomposition," *Computers & Operations Research*, vol. 127, p. 105127, 2021, doi:10.1016/j.cor.2020.105127.

[26] R. Roberti and M. Ruthmair, "Exact Methods for the Traveling Salesman Problem with Drone," *Transportation Science*, vol. 55, no. 2, pp. 315–335, 2021, doi:10.1287/trsc.2020.1017.

[27] B. Madani, M. Ndiaye, and S. Salhi, "Hybrid truck-drone delivery system with multi-visits and multi-launch and retrieval locations," *European Journal of Operational Research*, vol. 316, no. 1, pp. 100–125, 2024, doi:10.1016/j.ejor.2024.02.010.

[28] X. Ren, A. Froger, O. Jabali, and G. Liang, "A competitive heuristic algorithm for vehicle routing problems with drones," *European Journal of Operational Research*, vol. 318, no. 2, pp. 469–485, 2024, doi:10.1016/j.ejor.2024.05.031.

[29] H. Y. Jeong, B. D. Song, and S. Lee, "Truck-drone hybrid delivery routing: Payload-energy dependency and no-fly zones," *International Journal of Production Economics*, vol. 214, pp. 220–233, 2019, doi:10.1016/j.ijpe.2019.01.010.

[30] O. Dukkanci, B. Y. Kara, and T. Bektaş, "Minimizing energy and cost in range-limited drone deliveries with speed optimization," *Transportation Research Part C: Emerging Technologies*, vol. 125, p. 102985, 2021, doi:10.1016/j.trc.2021.102985.

[31] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions—I," *Mathematical Programming*, vol. 14, pp. 265–294, 1978, doi:10.1007/BF01588971.

[32] U. Feige, "A Threshold of $\ln n$ for Approximating Set Cover," *Journal of the ACM*, vol. 45, no. 4, pp. 634–652, 1998.

[33] V. Chvátal, "A Greedy Heuristic for the Set-Covering Problem," *Mathematics of Operations Research*, vol. 4, no. 3, pp. 233–235, 1979, doi:10.1287/moor.4.3.233.

[34] S. Khuller, A. Moss, and J. Naor, "The budgeted maximum coverage problem," *Information Processing Letters*, vol. 70, no. 1, pp. 39–45, 1999, doi:10.1016/S0020-0190(99)00031-9.