Minimizing Total Travel Time for Collaborative Package Delivery with Heterogeneous Drones

Thomas Erlebach
Department of Computer Science, Durham University, UK

Kelin Luo
Department of Computer Science and Engineering, University at Buffalo, NY, USA

Wen Zhang
Department of Computer Science and Engineering, University at Buffalo, NY, USA

Abstract

Given a fleet of drones with different speeds and a set of package delivery requests, the collaborative delivery problem asks for a schedule for the drones to collaboratively carry out all package deliveries, with the objective of minimizing the total travel time of all drones. We show that the best non-preemptive schedule (where a package that is picked up at its source is immediately delivered to its destination by one drone) is within a factor of three of the best preemptive schedule (where several drones can participate in the delivery of a single package). Then, we present a constant-factor approximation algorithm for the problem of computing the best non-preemptive schedule. The algorithm reduces the problem to a tree combination problem and uses a primal-dual approach to solve the latter. We have implemented a version of the algorithm optimized for practical efficiency and report the results of experiments on large-scale instances with synthetic and real-world data, demonstrating that our algorithm is scalable and delivers schedules of excellent quality.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Approximation algorithms, Primal-dual method, Heterogeneous pickup and delivery problem, Algorithm engineering

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.17

Related Version *Full Version (arXiv)*: `https://doi.org/10.48550/arXiv.2602.19535` [10]

Supplementary Material *Software*: `https://github.com/WenZhang-Vivien/Tree_construction_Primal_Dual_Routing` [9], archived at `swh:1:dir:9590a63eb16bd82996f44232019407698fc42911`

Acknowledgements This research was supported by data provided by Meituan.

1 Introduction

The rise of autonomous vehicle technology has profoundly transformed the logistics and delivery industry, enabling faster and more efficient package delivery. Given the varying serving speeds, consumption costs, and service areas of different vehicles or drones, one of the key challenges in this domain is optimizing the delivery process, particularly when multiple vehicles are available to complete pickup and delivery tasks [15]. This class of problems, known as Collaborative Delivery Problems (CD Problems), focuses on finding an optimal schedule to transport packages from one location to another through the collective effort of multiple vehicles. There are two modes in which vehicles can collaborate on completing the pickup and delivery tasks: In *coarse-grained* collaboration, the vehicles collaborate by partitioning the package delivery requests among themselves: Each package delivery request is assigned to one vehicle, and each of the vehicles independently follows a route based on its assigned requests. In *fine-grained* collaboration, multiple vehicles can participate in the fulfillment of a single request by handing over packages between them.

In this paper, we study the multi-package delivery problem in a collaborative setting. We extend the classical problem of multiple vehicle pickup to a setting where the available vehicles have different efficiencies, particularly in terms of speed. We formulate this problem as a Heterogeneous Collaborative Delivery Problem and investigate how delivery planning can be optimized to minimize the objective of total travel time. The objective of minimizing the total travel time directly reflects overall operational efficiency and cost. In contrast, minimizing the overall delivery completion time is a different optimization objective with a stronger emphasis on balancing workloads across drones, which we do not consider in this paper. We use the terms "drone" and "vehicle" interchangeably in the remainder of the paper.

Our contributions are: (1) We analyze the effectiveness of fine-grained collaboration among vehicles and show that the gap between serving packages with fine-grained collaboration and serving them with coarse-grained collaboration is bounded by a factor of 3. (2) Our main contribution is a constant-factor approximation algorithm for the Heterogeneous CD problem with coarse-grained collaboration. We reduce the problem to a Tree Combination Problem and adapt a primal-dual algorithm presented for the multiple-vehicle TSP to address it. Our model accounts for the heterogeneous capabilities of vehicles and achieves a constant approximation ratio for minimizing total travel time (or total cost). Furthermore, our reduction shifts the problem complexity to depend primarily on the number of vehicles rather than the number of requests, which is typically much larger than the number of vehicles. This advancement offers a practical framework for optimizing real-world routing problems, where variations in vehicle capabilities play a critical role. (3) We describe an optimized implementation of our algorithm and report experimental results regarding running time and solution quality for a range of synthetic and real-world instances. Compared to a baseline heuristic without approximation guarantee, our algorithm produces solutions of comparable and sometimes substantially better quality with a significantly reduced running time.

The remainder of the paper is structured as follows. Section 1.1 discusses related work. Section 2 introduces relevant notation and definitions and proves that the gap between the best fine-grained and coarse-grained schedules is bounded. In Section 3, we present our constant-factor approximation algorithm for the heterogeneous CD problem with coarse-grained collaboration. In Section 4, we describe the optimizations we have applied for the implementation of the algorithm. Then, Section 5 presents the experimental results. Missing proofs and other details omitted due to space restrictions can be found in the full version [10].

1.1 Related Work

The problem of delivering a single package using has been well studied [1, 2, 3]. Here, k agents with different velocities and energy consumption rates must collaboratively deliver a package from its source to its target in a given edge-weighted graph. It has been shown that one can in polynomial time compute a schedule that minimizes the total energy consumption [1] or the delivery time [2, 3]. If two packages must be delivered, minimizing the delivery time is NP-hard [3]. Collaborative delivery of a single package has also been studied in scenarios where each agent's travel distance is limited [4, 5], an agent's movement region is restricted [8], or where the package must travel along a given path [6].

The Dial-a-Ride Problem (DARP)–preemptive or non-preemptive–involves a set of objects (each specified by a source-destination node pair) and a fleet of vehicles. The objective is to compute the shortest possible tour such that all objects are picked up from their sources and delivered to their destinations, without exceeding the vehicle's capacity at any point in time. This problem is typically studied under capacity constraints, commonly for homogeneous

vehicles. Our problem can be viewed as a heterogeneous, unit-capacity, non-preemptive DARP. While a 1.8-approximation is known for the unit-capacity case on general graphs [11], the heterogeneous setting, to the best of our knowledge, remains unexplored.

A special case of our problem where the pickup location equals the drop-off location can be viewed as a Multiple Depot Heterogeneous Traveling Salesman Problem (MDHTSP), with only the minor difference that vehicles need to return to their depots in the MDHTSP. Rathinam et al. [14] present a 2-approximation algorithm for MDHTSP based on the primal-dual method [12]. Our algorithm also leverages the primal-dual method, extending their approach.

2 Preliminaries

We are given a metric space (X, d), where $d : X \times X \to \mathcal{R}_{\geq 0}$ is a distance function satisfying the triangle inequality. A set of vehicles (or drones) K with $k = |K|$ is specified, where each drone $j \in K$ has a depot (or initial location) $r_j \in X$ and a speed p_j. To traverse from node $u \in X$ to node $w \in X$, drone j takes time $\frac{d(u,w)}{p_j}$. We assume that for any two drones i and j with $i < j$, the speed satisfies $p_i \geq p_j$. We are also given a set of m packages to be delivered, each specified by a pickup location $s_i \in X$ and a drop-off location $t_i \in X$. We use the terms *package* and *s-t* pair interchangeably. Each drone can carry at most one package at a time.

As objective function we consider the total completion time, also known as the sum of completion times. For a given solution, the completion time is computed for every drone, and the objective function value is the sum of all these values.

We define a schedule π_j for drone j as an ordered (by pickup time τ) set of triples $\{(u_i, w_i, \tau)\}$, where the drone picks up package P_i at location u_i at time τ and delivers it to location w_i. For any two consecutive triples (u_i, w_i, τ) and $(u_{i'}, w_{i'}, \tau')$ in the schedule, the pickup time of the second must be at least the previous pickup time plus the travel time from u_i to w_i and from w_i to $u_{i'}$. The time τ of the first triple (u_i, w_i, τ) in π_j must be at least the time it takes the drone to travel from its depot r_j to u_i.

The schedule for each package P_i can be obtained by collecting all triples involving package i, i.e., $\bar{\pi}_i = \{(u_i, w_i, \tau), ...\}$, from the union of all drone schedules $\bigcup_j \pi_j$, and sorting them by the associated time τ. Note that for each package P_i, the first triple should have $u_i = s_i$, and the final triple should have $w_i = t_i$; these may be the same triple. A package can only be picked up at a location u_i if it is the source s_i, or after it has been dropped off at u_i by another drone (or the same drone) at an earlier time.

▶ **Definition 1** (Min-Sum Collaborative Delivery Problem (Min-Sum CD Problem)). *Given a metric space (X, d), a set of m packages $\mathcal{P} = \{s_i, t_i\}_{i=1}^m$ where s_i is the pick-up location and t_i is the drop-off location, and k drones with a depot r_j and speed p_j for each drone j, the goal is to find schedules (or routes) $\{\pi_j\}_{j\in[k]}$ such that the schedule for each drone j originates at its initial location r_j, and each package i is transported from its source s_i to its destination t_i. The objective is to minimize the total cost, defined as the sum of travel times across all drone routes $\sum_{j=1}^{k} \left(\tau_{f_j} + \frac{d(u_{f_j}, w_{f_j})}{p_j}\right)$, where $(u_{f_j}, w_{f_j}, \tau_{f_j})$ is the final triple in the schedule π_j of drone j. Note that drone j finishes its schedule at time $\tau_{f_j} + \frac{d(u_{f_j}, w_{f_j})}{p_j}$ because τ_{f_j} is the time at which it makes its final pick-up (at location u_{f_j}), and then its final move is to take the package P_{f_j} from u_{f_j} to w_{f_j} with speed p_j.*

We consider Min-Sum CD with both fine-grained and course-grained collaboration. In the remainder of the paper, we refer to these variants of the problem as the *preemptive* and the *non-preemptive* variant, respectively. In the *Preemptive Min-Sum CD* problem, after

picking up a package from its source, it may (repeatedly) be left at an intermediate location before being picked up by another (or the same) drone and transported further on its route to the destination. In this case, the schedule for each package may include multiple triples. In the *Non-preemptive Min-Sum CD* problem, a package, once picked up from its source, is carried by the drone until it is dropped off at its destination. In this case, the schedule for each package includes only a single triple (s_i, t_i, τ).

We first show that the impact of preemption on the Min-Sum CD problem is limited:

▶ **Theorem 2.** *An optimal schedule for the* Preemptive Min-Sum CD *problem with total cost* OPT *can be transformed into a non-preemptive schedule with cost at most* $3 \cdot \mathsf{OPT}$.

Proof Sketch. Consider an optimal schedule Π^* for the *Preemptive Min-Sum CD* problem, with π_i^* for $i \in [k]$ denoting the schedule of drone i in the optimal schedule. We construct a feasible schedule for the Non-preemptive Min-Sum CD problem with cost at most $3 \cdot \mathsf{OPT}$. For this, we process the routes π_i^* in order of increasing i. When processing π_i^*, consider all packages P_j that are carried at least once by π_i^* but were not carried by any previously processed route. If π_i^* transports P_j from u_j to w_j, we change the route and let drone i first move from u_j to s_j to pick up P_j at its source, then carry the package to t_j to delivery it, and finally return to w_j and continue the original route π_i^*. Note that the increase in the length of the route π_i^* due to this change for package P_j is at most twice the length len_j of the path along which P_j is transported from s_j to t_j by Π^*. The total increase in the travel time of all routes is therefore bounded by $\sum_{j \in [m]} \frac{2\mathrm{len}_j}{p_{i(j)}}$, where $i(j)$ is the fastest drone involved in transporting P_j in Π^*. This increase is at most $2\mathsf{OPT}$, and as the schedule Π^* has cost OPT, the final cost of the resulting non-preemptive schedule is at most $3\mathsf{OPT}$. ◀

▶ **Corollary 3.** *An α-approximation algorithm for the* Non-preemptive Min-Sum CD *problem implies a 3α-approximation algorithm for the* Preemptive Min-Sum CD problem.

3 Non-preemptive Min-Sum CD

In this section we present a constant-factor approximation algorithm for the Non-preemptive Min-Sum CD problem. By Theorem 2, the same algorithm is also a constant-factor approximation algorithm for the Preemptive Min-Sum CD problem.

We translate the Min-Sum CD problem into a *Tree Combination Problem* where trees from a set of *original trees* can be combined into *large trees* by adding edges between them:

▶ **Definition 4** (Tree Combination Problem)**.** *Given a metric space (V, d) and a set of trees $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$, where $V(T_i) \subseteq V$ and each tree T_i is rooted at a drone location $r_i \in V$ with a corresponding drone speed p_i, the objective is to combine trees so as to minimize the sum, over all the resulting large trees, of the total edge length of the large tree divided by the highest speed among the drones associated with the original trees included in the large tree.*

▶ **Lemma 5** (Reducing Min-Sum CD to a Tree Combination Problem)**.** *Every given instance of the Min-Sum CD problem can be reduced in polynomial time to an instance of the Tree Combination Problem such that an α-approximation algorithm for the Tree Combination Problem implies a 6α-approximation algorithm for the Min-Sum CD problem.*

Proof Sketch. Given an instance of the Min-Sum CD problem, we construct an instance of the Tree Combination Problem with the following algorithm (Algorithm 1):

A solution to the constructed instance of the Tree Combination Problem with objective value x can be used to construct a solution to the Non-preemptive Min-Sum CD problem with sum of completion times at most $2x$ as follows: For each large tree T' with highest drone speed p_i, let the drone with speed p_i traverse an Euler tour of T'.

Algorithm 1 Constructing Trees $((V,d),\{s_i,t_i\}_{i\in[m]},\{r_i\}_{i\in[k]})$.

1: Create a new graph G' from the metric $(\{s_i\}_{i\in[m]}\cup\{r_i\}_{i\in[k]}),d)$ by contracting the set of depots $R=\{r_1,r_2,\ldots,r_k\}$ into a single node r.
2: Compute the minimum spanning tree (MST) T of the graph G'.
3: Obtain the forest $\{T_i\}_{i\in[k]}$ from T by un-contracting the node r back into the original depots in R, such that each tree T_i in the forest is rooted at a distinct depot r_i.
4: For each package source vertex s_i, connect vertex t_i to s_i in its respective tree.

In the full paper [10], we show that a solution with cost OPT for the Non-preemptive Min-Sum CD problem can be used to construct a solution with cost at most 3OPT for the Tree Combination Problem. An α-approximation algorithm for the Tree Combination Problem therefore produces a solution of cost at most 3αOPT, which then gives a solution of cost at most 6αOPT for the Non-preemptive Min-Sum CD problem. ◀

Our problem can thus be reduced to solving the Tree Combination Problem, where the input trees are the outputs of Algorithm 1. Without loss of generality, the input consists of k trees, denoted as $T_1,T_2,\ldots,T_k$, where $w(T_i)$ represents the total length of the tree T_i. Each tree T_i is rooted at a drone location r_i with an associated drone speed p_i.

We now present a 2-approximation algorithm for the Tree Combination Problem. Inspired by the study on the Heterogeneous Traveling Salesman Problem [14], we employ a multi-level primal-dual algorithm. If vehicles travel at different speeds and the travel cost is defined as the ratio of the traveled distance to the vehicle's speed, then the travel cost adheres to the principle of monotone costs, which is a prerequisite for the approach to apply. We model the solution to include penalties for trees not visited by the highest-speed vehicle.

3.1 LP Relaxation and Its Dual

Consider the Tree Combination Problem involving k trees represented as vertices. We categorize these vertices into $h\le k$ distinct levels based on speed, organized from fastest to slowest, with level 1 being the fastest. Level $l\in[h]$ contains h_l vertices. Let $v_{l,i}$ represent the i-th vertex at speed level l. The set of vertices at level l is denoted as $V_l=\{v_{l,i}\}_{i\in[h_l]}$. For each vertex $v_{l,i}$, where $l\in[h]$ and $i\in[h_l]$, the weight is defined as $w(v_{l,i})$, the set of vertices in the original tree it represents is $V(v_{l,i})$, and its speed is denoted as p_l. Note that here we reuse the notation $p_1,\ldots,p_h$ to denote the distinct speed values (i.e., the speed-equivalence levels), rather than individual drone speeds as introduced in Section 2.

The input to the Tree Combination Problem is a graph with vertex set $V=\bigcup_l V_l$, edge set $E=\left\{(u,v)\mid u,v\in\bigcup_{l\in[h]}V_l\right\}$, where the length of an edge between two nodes $v_{l,i},v_{l',i'}$ is given by $\ell(v_{l,i},v_{l',i'})=\min_{a\in V(v_{l',i'}),\ b\in V(v_{l,i})}d(a,b)$. A resulting tree rooted at $v_{l,i}$ (the vertex with highest speed in the tree) is defined by $\tilde{T}(l,i)=(\tilde{V}_{l,i},\tilde{E}_{l,i})$ with $\tilde{V}_{l,i}\subseteq V$ and $\tilde{E}_{l,i}\subseteq E$. The total cost of this tree is equal to the sum of edge costs $\sum_{(u,v)\in\tilde{E}_{l,i}}\ell(u,v)/p_l$ plus the sum of vertex costs $\sum_{v\in\tilde{V}_{l,i}}w(v)/p_l$ if $|\tilde{V}_{l,i}|\ge 1$, and is equal to 0 otherwise. (We set $\tilde{V}_{l,i}=\tilde{E}_{l,i}=\emptyset$ if none of the resulting trees is rooted at $v_{l,i}$.) The objective is to identify a set of trees such that each vertex is included in exactly one of the resulting trees, while minimizing the total cost, i.e.,

$$\sum_{l\in[h],i\in[h_l]}\left(\sum_{(u,v)\in\tilde{E}_{l,i}}\ell(u,v)/p_l+\sum_{v\in\tilde{V}_{l,i}}w(v)/p_l\right).$$

Figure 1 illustrates a Tree Combination Problem involving nine trees across three levels.

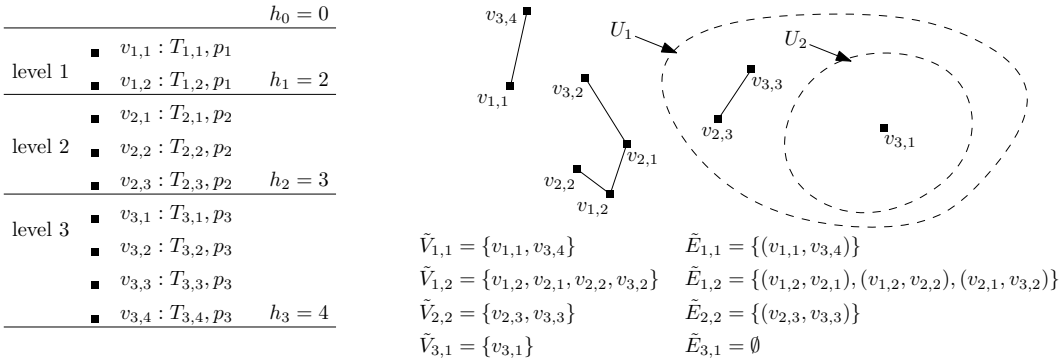


Figure 1 Instance of the Tree Combination Problem with nine vertices, each associated with one of three different speeds $p_1 > p_2 > p_3$. Illustration of a feasible solution with four large trees.

We design an approximation algorithm using the primal-dual method. The vertex set $\bar{V}_l = \bigcup_{l'>l} V_{l'}$ consists of vertices that may appear in the resulting trees rooted at a vertex in level l, but are initially in a level larger than l. The edge set $\bar{E}_l = \{e \in E \mid \text{both endpoints of } e \text{ have level } \geq l\}$ consists of edges that may appear in a resulting tree rooted at a vertex in level l. We formulate an (integer) LP using the following variables:

- Variables x_e^l indicate whether an edge $e \in \bar{E}_l$ is included in the resulting tree rooted at a vertex in V_l, where $l \in \{1, \ldots, h-1\}$.
- Variables z_U^l are defined for $l = 1, \ldots, h-1$ and for any subset $U \subseteq \bar{V}_l$, with $z_U^l = 1$ if U represents the set of all vertices with speed $p_{l+1}, p_{l+2}, ..., p_h$ that do not get connected to a vertex with speed p_l or faster.

Denote the cost of traversing an edge $e \in \bar{E}_l$ by its root vertex (drone) with speed p_l by $\mathsf{cost}_l(e) = \ell(e)/p_l$, and the cost of serving a vertex v by its root vertex (drone) with speed p_l as $w_l(v) = w(v)/p_l$. Let $\pi_l(U) = \sum_{v \in U}(w_{l+1}(v) - w_l(v))$ be the level-l penalty if all vertices in $U \subseteq \bar{V}_l$ are served by a root vertex with speed less than or equal to p_{l+1}. Notice that if a vertex v is served by a root vertex with speed p_l, then it will be included in the sets $U_1, U_2, \ldots, U_{l-1}$, and the total penalty incurred for this vertex will be $\pi(\{v\}) = \sum_{i=1}^{l-1} \pi_i(\{v\}) = w_l(v) - w_1(v)$. The ILP is then:

$$\text{Min} \quad \sum_{l \in [h-1]} \sum_{e \in \bar{E}_l} \mathsf{cost}_l(e) x_e^l + \sum_{l \in [h-1]} \sum_{U \subseteq \bar{V}_l} \pi_l(U) z_U^l \tag{1}$$

$$\text{st.} \quad \sum_{e \in \delta_l(S)} x_e^l \geq \sum_{U: S \subseteq U \subseteq \bar{V}_{l-1}} z_U^{l-1} - \sum_{U: S \subseteq U \subseteq \bar{V}_l} z_U^l \quad \forall S \subseteq \bar{V}_l, l \in [h-1], \tag{2}$$

$$z_V^0 = 1; \quad z_U^0 = 0 \quad \forall U \neq V, \tag{3}$$

$$x_e^l \in \{0,1\} \quad \forall e \in \bar{E}_l, l \in [h-1], \tag{4}$$

$$z_U^l \in \{0,1\} \quad \forall U \subseteq \bar{V}_l, l \in [h-1]. \tag{5}$$

Here in constraint (2), we let $\delta_l(S)$ denote the set of edges in $\bar{E}_l$ with exactly one endpoint in S, i.e., $\delta_l(S) = \delta(S) \cap \bar{E}_l$. If we consider solutions where, for each l, $z_U^l = 1$ for exactly one U and $z_U^l = 0$ for all other U, then both terms $\sum_{U: S \subseteq U \subseteq \bar{V}_{l-1}} z_U^{l-1}$ and $\sum_{U: S \subseteq U \subseteq \bar{V}_l} z_U^l$ in constraint (2) only take binary values, either 0 or 1. Then this constraint is trivially satisfied except for the case when $\sum_{U: S \subseteq U \subseteq \bar{V}_{l-1}} z_U^{l-1} = 1$ and $\sum_{U: S \subseteq U \subseteq \bar{V}_l} z_U^l = 0$. But this case can occur only if there is at least one vertex in S that must be connected to a tree rooted at level l. The constraint reduces to $\sum_{e \in \delta_l(S)} x_e^l \geq 1$ in that case and it is ensured that each

such vertex is indeed connected to a tree rooted at level l. The IP formulation also allows solutions in which, for some l, $z_U^l = 1$ for more than one set U, but this cannot increase the objective value of an optimal solution. The objective is to minimize the total cost, including both connection (edge) costs and vertex service costs. If a vertex v is connected to a tree rooted at level l, then its cost (penalty) is accounted for in every preceding level from 1 to $l-1$. These costs are included in the penalties associated with the sets $U_1, U_2, \ldots, U_{l-1}$.

If we replace the integrality conditions $x_e^l \in \{0,1\}$ and $z_U^l \in \{0,1\}$ with $x_e^l \geq 0$ and $z_U^l \geq 0$ and take the dual of the linear program, we obtain the following:

$$\text{Max} \quad \sum_{S \subseteq \bar{V}_1} Y_1(S) \tag{6}$$

$$\text{st.} \quad \sum_{S \subseteq \bar{V}_l : e \in \delta_l(S)} Y_l(S) \leq \mathsf{cost}_l(e), \quad e \in \bar{E}_l, \forall l \in [h-1], \tag{7}$$

$$\sum_{S : S \subseteq U} Y_l(S) - \sum_{S : S \subseteq U - V_{l+1}} Y_{l+1}(S) \leq \pi_l(U) \quad \forall U \subseteq \bar{V}_l, \forall l \in [h-1], \tag{8}$$

$$Y_l(S) \geq 0 \quad \forall S \subseteq \bar{V}_l, l \in [h-1]. \tag{9}$$

The constraints (7) pertain to edges, whereas the constraints (8) are related to sets of vertices.

3.2 Primal-Dual Algorithm

The algorithm maintains a separate forest for each level: The forest of a level consists of trees rooted at vertices within that level to which vertices from higher levels are connected, and possibly some trees that consist only of vertices from higher levels. This organization is achieved through the application of the primal-dual moat-growing procedure, which is implemented independently and simultaneously across each level's graphs. We define $F_l(t)$ to be the resulting forest at level $l \in [h]$ in the graph $(V_l \cup \bar{V}_l, \bar{E}_l)$ by the end of iteration t of the main loop. For any two levels l and l', where $l < l'$, and at any iteration t, consider a component C_i in forest $F_l(t)$ and C_j in forest $F_{l'}(t)$. Then C_i is considered an ancestor of C_j, and C_j a descendant of C_i, if $C_i \supseteq C_j$.

▶ **Definition 6** (Remaining Potential). *For a given set of vertices $U \subseteq \bar{V}_l$ and a dual feasible solution $Y_l(U)$, we say that $P(Y_l(U), \pi) = \pi_l(U) - (\sum_{S:S\subseteq U} Y_l(S) - \sum_{S:S\subseteq U-Y_{l+1}} Y_{l+1}(S))$ is the remaining potential of the set U in the l-th level.*

A component is considered **active** at the start of an iteration if it consists of higher-level vertices and its dual variable is allowed to increase during the iteration without violating any constraints in the dual problem. A component is deemed **inactive** if it contains a vertex at the corresponding level or if it is a descendant of an inactive component. A component is **frozen** if it has ceased growing due to constraint (8), at which point its remaining potential becomes 0. If a component is inactive or frozen at the start of an iteration, its dual value will not change during that iteration. When a component becomes frozen, meaning its remaining potential is reduced to zero, it implies that each of its descendants at the next higher level has either already become frozen or has become inactive.

While the dual LP has only variables $Y_l(S)$ for $S \subseteq \bar{V}_l$, $l \in [h-1]$, in our description of the primal-dual algorithm we consider variables $Y_l(S)$ for $S \subseteq V_l \cup \bar{V}_l$, $l \in [h]$ for ease of presentation. The values of the additionally introduced variables are zero at all times.

Initially, each forest $F_l(0)$ for level $1 \leq l \leq h-1$ at time 0 consists of singleton components, each of which is a vertex of $V_l \cup \bar{V}_l$. The singleton components containing only vertices from higher levels $\bar{V}_l$ are considered active, while any component containing a vertex with speed p_l is deemed inactive. The dual variables corresponding to all active and inactive components are initialized to zero: For every $l \in [h]$ and $v \in V_l$, we have $Y_{l'}(\{v\}) = 0$ for all $l' \in [l]$.

The *main loop* of the primal-dual algorithm consists of iterations in which the dual variables of all active components in all forests are simultaneously increased by the same amount as much as possible until at least one of the constraints in the dual problem becomes tight. If constraints in (7) become tight, then a tight constraint corresponding to the vertex with the lowest level (denoted as l) is chosen first. The edge corresponding to the chosen constraint is added to the forest, merging two components C_1 and C_2. If both C_1 and C_2 do not contain a vertex with speed p_l, then the merged component remains active. If one of the components contains a vertex with speed p_l (say, vertex $v_{l,j} \in C_2$), then the merged component and all its active descendants become inactive. If a constraint (8) becomes tight for some component, i.e., $P(Y_l(U), \pi) = 0$, then the corresponding component U in level l is deactivated and becomes frozen. The main loop of the algorithm terminates when every component in all forests is either inactive or frozen. See Figure 2 for an illustration.

▶ **Lemma 7.** *The main loop terminates in* $\sum_{l \in [h-1]} |V_l| \cdot (2l-1)$ *iterations.*

Note that the number of iterations of the main loop depends only on k and h, but not on n. When the main loop of the primal-dual procedure has terminated, a pruning procedure is carried out that selects a sub-forest $\tilde{F}_l$ of each level l forest F_l such that each tree of $\tilde{F}_l$ is rooted at one of the vertices in V_l and each vertex is connected in exactly one of the resulting sub-forests. We refer to the full version [10] for details of the pruning procedure and a proof that the algorithm produces a feasible solution to the Tree Combination Problem.

Approximation Guarantee

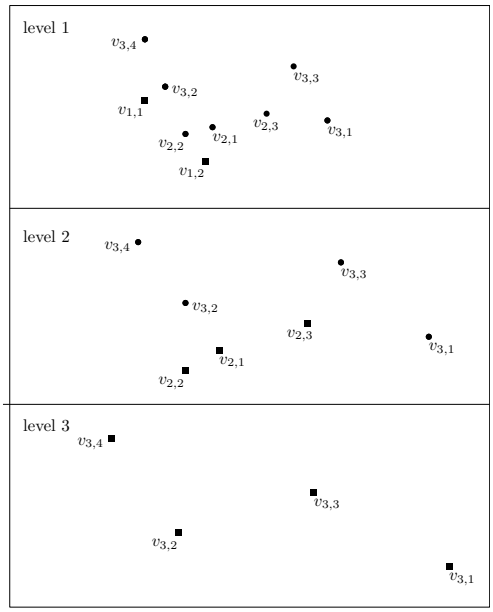
We can prove that the cost of the ILP solution produced by the primal-dual algorithm is at most twice the optimal cost. For this, we show that the dual value of the LP relaxation can be equivalently written as the sum of the dual values and penalty costs corresponding to vertices at each level. This then allows us to bound the cost of the edges in the forest of each level with respect to the dual value. For details, we refer to the full version [10].

The cost of the constructed solution to the Tree Combination Problem equals the cost of the solution to the ILP obtained via the primal-dual algorithm plus the non-negative fixed cost $\sum_{v \in V} w(v)/p_1$. Therefore, we get a 2-approximation algorithm for the Tree Combination Problem with multiple different speeds. Combining this result with Lemma 5, we get a 12-approximation algorithm for the Non-preemptive Min-Sum CD problem.

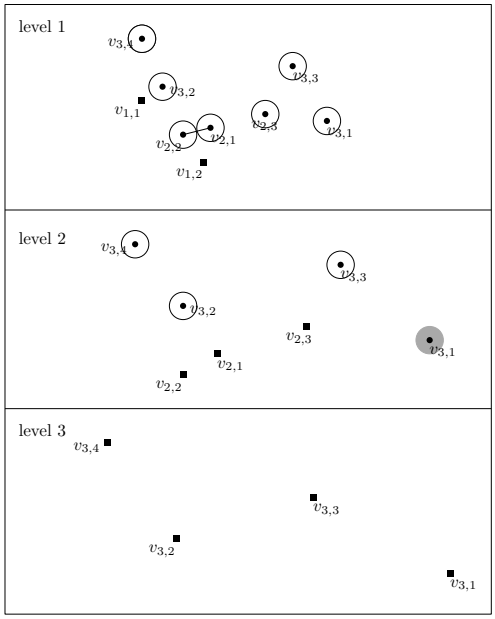
▶ **Theorem 8** (Min-Sum CD). *There exists an* $O(1)$*-approximation algorithm for the Non-preemptive Min-Sum CD problem.*

4 Algorithmic Adjustments for Implementation

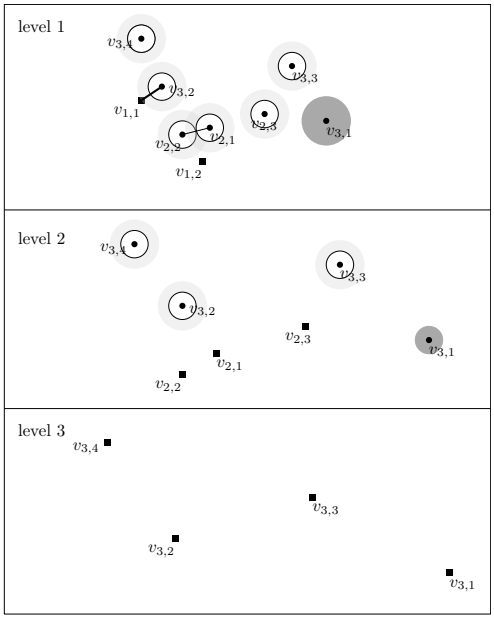
Before presenting the experimental results, we discuss two implementation refinements that improve both running time and solution quality. First, we refine the algorithm for tree construction (Algorithm 1). A straightforward implementation of step 2 of Algorithm 1 would involve constructing the graph G' of size $O(n^2)$ and computing an MST in $O(n^2)$ time. As we only consider instances where all input points lie in the Euclidean plane in our experiments,



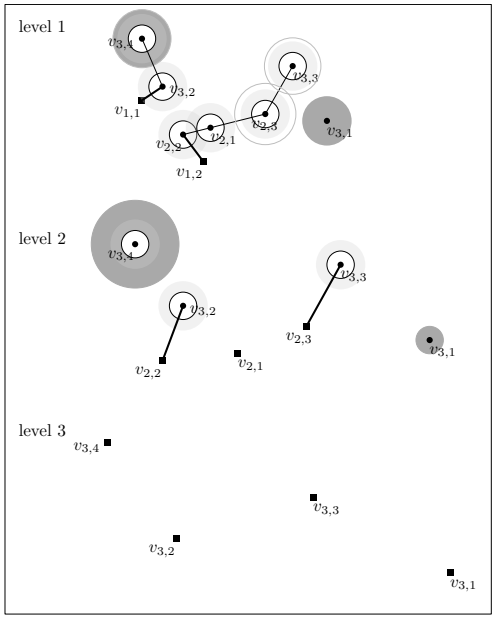
(a) Primal-dual: Initialization
For each level $l \in [3]$, the vertices V_l are inactive and represented by squares, while the other vertices $\bar{V}_l$ are active and represented by disks. In each iteration, all active components' dual variables are increased equally until a dual constraint (either edge constraint or potential constraint) becomes tight.



(b) Example of two tight constraints:
Edge constraint: $Y_1(v_{2,1}) + Y_1(v_{2,2}) = \text{cost}_1(\{v_{2,1}, v_{2,2}\})$, thus edge $\{v_{2,1}, v_{2,2}\}$ is added to E_1, the components $\{v_{2,1}\}$ and $\{v_{2,2}\}$ in level 1 become inactive and the merged component $\{v_{2,1}, v_{2,2}\}$ becomes active. The component $v_{3,1}$ in level 2 becomes frozen.



(c) Edge $\{v_{3,2}, v_{1,1}\}$ is added to E_1, component $v_{3,1}$ in level 1 becomes frozen.



(d) Final iteration. All components become inactive or frozen.

Figure 2 Illustration of the primal-dual algorithm.

to speed up the computation we first compute a Delaunay triangulation of the sources and depots in $O(n \log n)$ time [7] before contracting the depots into a single depot r (referred to as *super-depot*). Furthermore, as vehicles move from the target of one request to the source of another, we found that target-to-source distances are better suited than source-to-source distances for the generation of a good spanning tree. We use the location of a target in the Delaunay triangulation to determine for each target a set of nearby candidate sources (see the full version [10] for details). Using a Prim-like procedure to generate the spanning tree, whenever we add an s-t pair to the tree, we update the priorities of the nearby candidate sources of the target t as well as the neighbors of s in the Delaunay triangulation. We can then build the tree by always adding the request whose source has the minimum priority with respect to target-to-source distance. Alternatively, we add the request with minimum target-to-source distance d_{ts} only if d_{ts} is at most k times the minimum source-to-source distance d_{ss} between the tree and a request not in the tree, where k is a small constant. This latter variant ensures that the cost of the spanning tree computed is within a constant factor of the MST of sources (with targets attached). We add "MST k" in brackets after the name of an algorithm to indicate that this variant is used.

Second, we consider three alternatives for converting the large trees into tours. Our first implementation performs a DFS of each large tree and lets the vehicle serve the requests in the order in which their sources are first visited by the DFS. The length of a tour constructed in this way is within a constant factor of the total length of the large tree (as needed in the theoretical approximation analysis). We refer to the algorithm with this tour construction method as *Primal–Dual algorithm with Depth-First Search Routing* (PD-DFS). It runs in $O(n)$ time.

As an alternative, we implemented the following cheapest-insertion heuristic: Build the tour for a large tree incrementally. Initially, it consists only of the depot of the fastest vehicle. Then process the requests in arbitrary order and insert each request at the position where it leads to the smallest increase in length (either right after the depot or right after the target of a request already in the tour). We refer to this method as PD-Greedy. It runs in $O(\sum N_i^2)$ time, where N_i is the number of requests in the i-th large tree.

As a final alternative, we implemented a two-stage version of the cheapest-insertion heuristic: First, use the cheapest-insertion heuristic to build a tour for each of the original trees included in the large tree. Then, use the cheapest insertion heuristic again to produce an order of the tours produced for the original trees (discarding their depots). We refer to this "double greedy" method as (PD-DGreedy).

We note that PD-DFS (MST k) is guaranteed to produce a solution that is a constant-factor approximation of the optimal solution, while the other algorithm variants do not guarantee this property for all inputs. For the MST k modification, we found in our experiments that the choice $k = 7$ worked best. The algorithm variants using this modification are referred to as PD-DFS (MST 7), PD-Greedy (MST 7), and PD-DGreedy (MST 7).

5 Computational Experiments

5.1 Synthetic Datasets

We consider three synthetic datasets in our experimental evaluation. The first dataset is constructed based on a worst-case instance (see the full version [10]) for the baseline algorithm with which we compare the performance of our algorithms. It is used to demonstrate the guaranteed performance of our algorithms relative to the baseline. The second dataset, denoted by SY-U, consists of instances in which all locations–including request pickup

locations, drop-off locations, and depot locations–are independently generated from a uniform distribution over a $[0, 100] \times [0, 100]$ grid. The third dataset, denoted by SY-GMM, is generated using a Gaussian mixture model with a number of clusters. The results for SY-GMM are generally similar to those for SY-U and can be found in the full version [10].

Table 1 Parameter settings for the synthetic datasets. Bold values indicate fixed parameters used in the sensitivity analyses. n is the number of requests, k the number of depots, and h the number of speeds/levels. The speed at level 1 is 50 and decreases by 5 at each subsequent level.

Parameter	Worst-case	SY-U
n	$5, 10, 15, 20\ (\times 10^3)$	$5, \mathbf{10}, 15, 20\ (\times 10^3)$
k	2	$1, 3, 5, 7, 9, 10, 15, 20, \mathbf{30}, 40\ (\times 3)$
h	2	$1, 2, \mathbf{3}, 4, 5$

5.2 Real-World Datasets

We also conduct experiments using a real-world dataset collected from Meituan [13]. The dataset contains orders, couriers (we treat the initial locations of the couriers as depots), and delivery records from an anonymized midsize city in China. To capture both regular and peak demand patterns, we select one representative weekday (October 18, 2022) and one representative weekend day (October 22, 2022) for evaluation. To study scalability under varying system sizes, we consider multiple depot-scale settings. For each setting, depots are randomly sampled from the full pool of available depots. For each depot configuration and selected day, all orders recorded on that day are used as requests. Each location is represented by its latitude and longitude. The distance between two locations is computed using Euclidean distance. A summary of key statistics for all experimental settings is reported in Table 2.

Table 2 Parameter settings for the real-world datasets.

Parameter	Weekday	Weekend
n (number of requests)	69,103	77,638
k (number of depots)	$1, 3, 5, 7, 9, 10, 15, 20, 30, 40\ (\times 3)$	
h (number of speed levels)	3	

5.3 Baseline Algorithm

We are not aware of any previously proposed methods for the heterogeneous collaborative delivery problem. Therefore, we consider the following cheapest-insertion heuristic algorithm as the baseline in our experiments: The algorithm begins with an arbitrary order of requests and processes them sequentially. For each request, it evaluates all vehicles and all insertion positions within their current routes (a request can be inserted at the beginning, i.e., as the first request to be served after the vehicle leaves the the depot, or after the completion of the delivery of a package that is already contained in the route), computing the additional travel time incurred. The request is then assigned to the vehicle and insertion position that yields the minimum increase in cost. This process is repeated until all requests are placed, producing a feasible set of routes that serves as a baseline for comparison in our experiments. The running time of the baseline algorithm is $O(n(k + n))$.

We emphasize that this greedy cheapest-insertion heuristic is not a constant-factor approximation, even in the special case where each request is of the form $s_i = t_i$. In the full version [10] we show how to construct arbitrarily large instances where the algorithm incurs a cost $\Theta(\alpha)$ times the optimal solution, where $\alpha = v_{\max}/v_{\min}$ is the ratio of the maximum to minimum vehicle speed.

5.4 Experimental Evaluation

All experiments were conducted on a laptop with Apple M2 Pro processor and 16 GB main memory running macOS 15.2. Our implementations, which allow all experimental results to be fully reproduced, are publicly available at `https://github.com/WenZhang-Vivien/Tree_construction_Primal_Dual_Routing`.

5.4.1 Computational Results in Synthetic Data Sets

In this section, we present the results on the worst-case and SY-U datasets. For the total travel time objective, we find that the baseline algorithm performs well in synthetic data sets, but is slowest, and in some instances can perform significantly worse due to the lack of a performance guarantee. All three PD-based algorithms run faster than the baseline algorithm. Among them, PD-Greedy consistently achieves performance close to that of the baseline algorithm. PD-DFS is the fastest algorithm, but its solution quality in the experiments is worse than that of the other two. PD-DGreedy offers a good trade-off: it is faster than PD-Greedy while achieving better performance than PD-DFS.

5.4.1.1 Computational Results in Worst-Case Data Sets

We construct a worst-case instance (for the baseline algorithm) on a line metric with two speed levels, as described in [10]. Specifically, we set ε (an extra parameter of the construction [10]), $v_{\max}$, and $v_{\min}$ as fixed parameters and increase the number of source–target pairs n. We let $\alpha = v_{\max}/v_{\min} = 1,000$. Figure 3 shows the solution cost and running-time versus the number n of requests. As n increases, the total cost of the baseline algorithm grows with n^2 whereas the costs of all PD-based algorithms are about $1,000$ times smaller, demonstrating the superior performance of our proposed methods. The MST 7 variants of the PD algorithms produce the same solutions as the variants not using the MST 7 modification and are not shown. Compared to the other algorithms, the running time of PD-DFS grows much more slowly as the number of requests increases.

5.4.1.2 Computational Results in SY-U Data Set

Figures 4, 5, and 7 illustrate the impact of the number of source–target pairs n, the number of depots k, and the number of speeds/levels h on algorithm performance for data generated from a uniform distribution.

For a fixed number of depots ($k = 90$) and a fixed number of speed levels ($h = 3$), as the number of source-target pairs n increases from 5×10^3 to 20×10^3 as shown in Figure 4, the ratio between the total travel time objective of our PD-Greedy (and PD-DGreedy) algorithm and that of the baseline algorithm decreases and approaches 1. Meanwhile, the ratio of the running times remains below 1 and continues to decrease, indicating a growing computational advantage of our approach at larger scales. The MST 7 variants of the PD algorithms tend to produce slightly better solutions at the expense of a slightly increased running time.

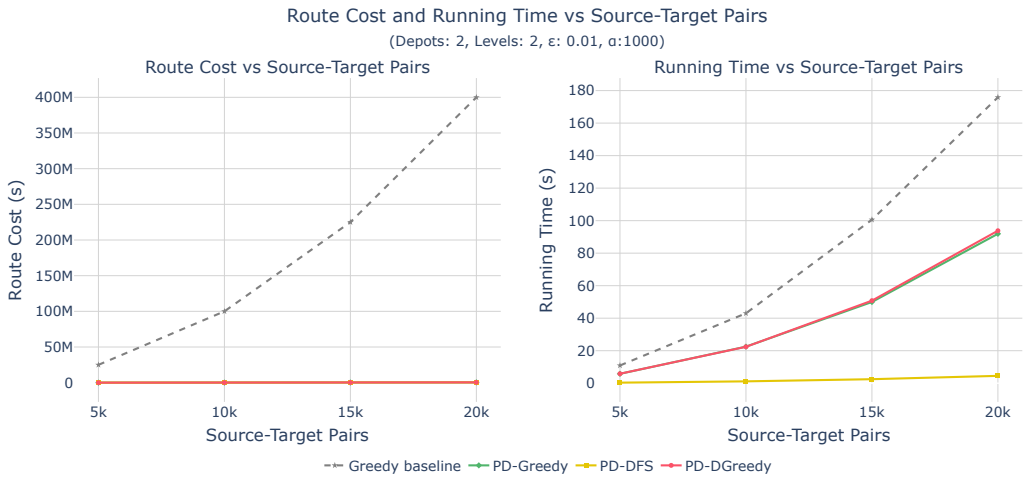


Figure 3 Results on the worst-case instance for the baseline algorithm with varying n, where $\alpha = v_{\max}/v_{\min}$ and ϵ is a small constant parameter.

Figure 6a illustrates the breakdown of the running time of the individual components of each PD-based algorithm as the number of request pairs increases. When n is small (e.g., $n = 5{,}000$), the primal–dual component dominates the running time. As n increases, the routing phases account for a larger portion of the total running time.

For a fixed number of source-target pairs ($n = 10 \times 10^3$) and a fixed number of speed levels ($h = 3$), when the number of depots k is small (less than 30), as shown in Figure 5, our PD-Greedy (and PD-DGreedy) algorithm outperforms the baseline in terms of solution quality; with an increasing number of depots, the baseline algorithm attains slightly better performance. In terms of running time, the PD-based algorithms are always faster than the baseline. Interestingly, the running time of all PD-based algorithms decreases as the number of depots increases when the number of depots is relatively small (less than 60), and then increases thereafter. This behavior can be explained by the changing dominance of different algorithmic components, see the breakdown shown in Figure 6b. These observations suggest that, for a fixed number of source-target pairs, there may exist an appropriate number of depots that simultaneously yields best solution quality and fast running time.

For a fixed number of source-target pairs ($n = 10 \times 10^3$) and a fixed number of depots ($k = 90$), as the number of speed levels h increases, as shown in Figure 7, the performance of our PD-Greedy algorithm is slightly worse than that of the baseline algorithm when the number of speed levels is small (less than 3), but becomes superior as the number of speed levels increases. The running time of the baseline algorithm remains largely unchanged as the number of speed levels increases, but the running time of our PD-based algorithms increases with the number of speed levels (due to an increase of the number of iterations of the Primal-Dual component), but they consistently remain faster than the baseline algorithm.

5.4.2 Computational Results in Realworld Data

As observed in the synthetic datasets, a relatively small number of depots (vehicles) is sufficient to achieve good performance with low running time. Accordingly, we focus on representative settings with different numbers of depots k, selected at random, and compare

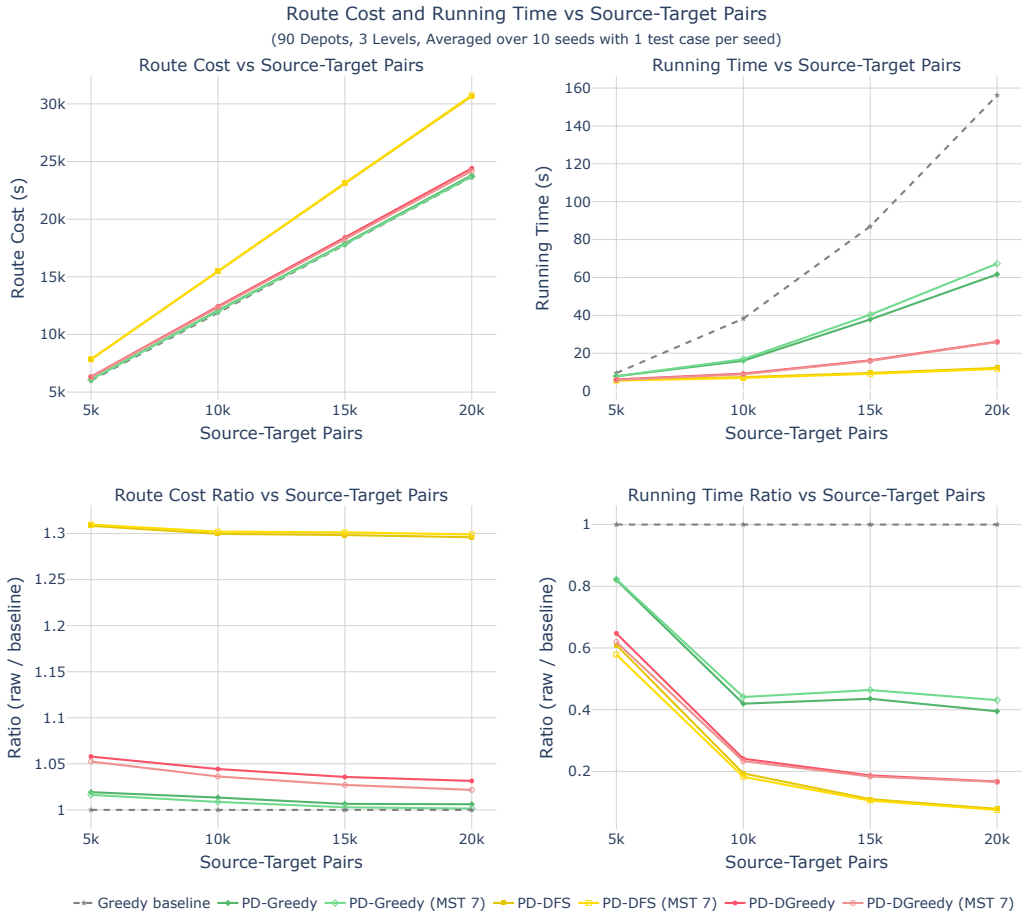


Figure 4 Results on the Uniform synthetic dataset (SY-U) with varying n.

the performance of the three PD-based algorithms against the baseline algorithm on both weekday and weekend real-world data, considering both the full set of depots (only for the baseline algorithm) and randomly selected depot subsets.

From Figure 8 (weekend data), we observe trends consistent with the synthetic experiments. (The results for weekday data are similar and can be found in the full version [10].) PD-Greedy achieves the best solution quality, while PD-DFS is the fastest. As the number of depots increases, the total route cost of the PD-based algorithms – particularly PD-Greedy and PD-DGreedy – slightly increases but remains within a factor of 1.1 of the baseline. Meanwhile, their running time decreases due to fewer requests per depot. This trend becomes more pronounced for large-scale real-world source–target request pairs.

We remark that, for both the synthetic datasets and the real-world datasets, the effects of varying n, k, and h exhibit similar trends. As the number of request pairs increases, our algorithm continues to achieve favorable performance, particularly when fewer depots are used, while maintaining relatively low running times.

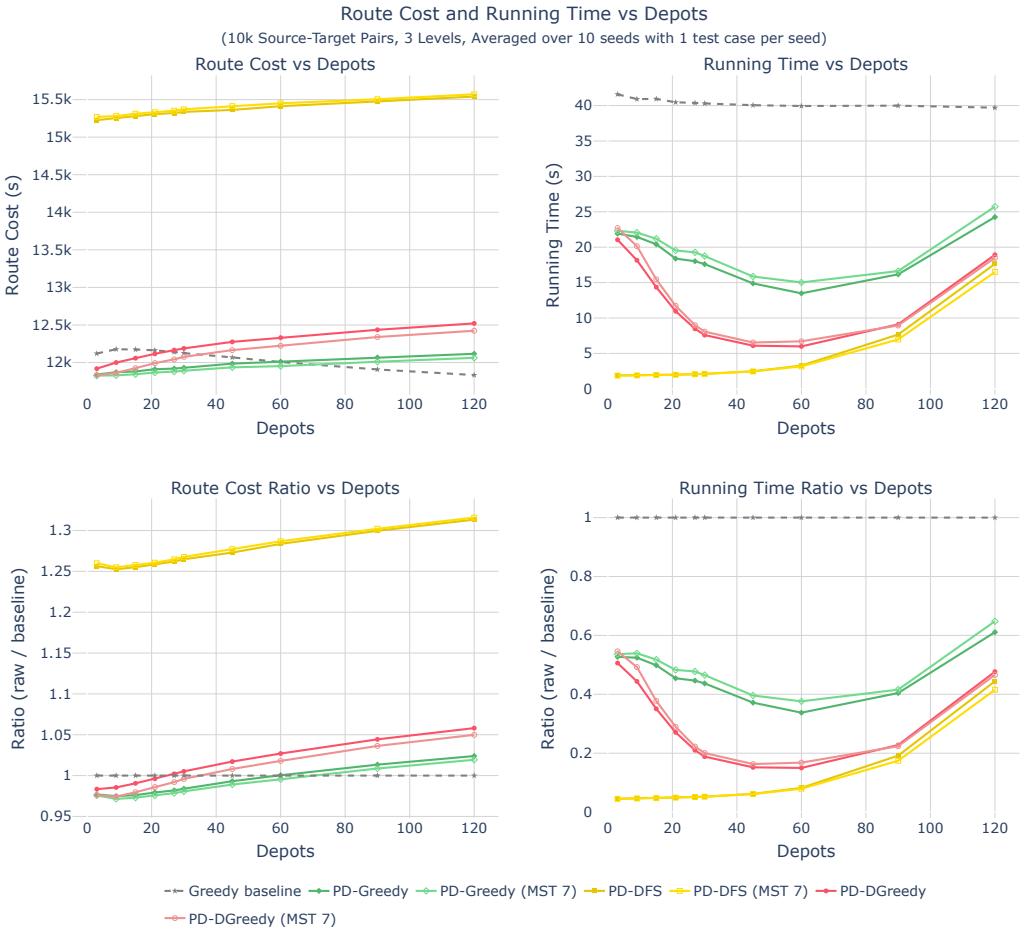
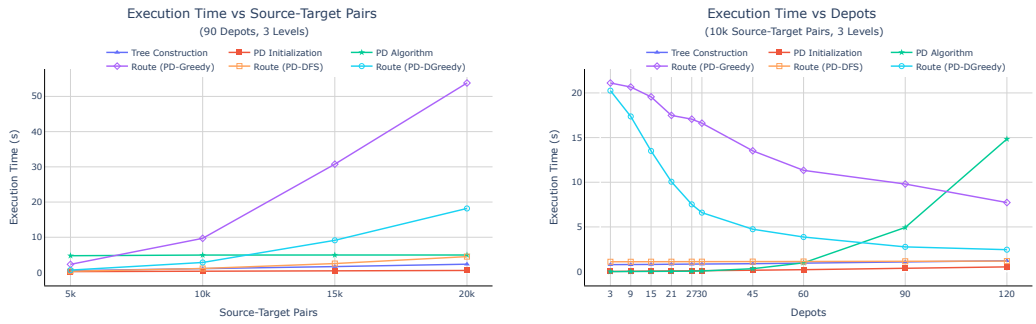


Figure 5 Results on the Uniform synthetic dataset (SY-U) with varying k.



(a) Varying n. **(b)** Varying k.

Figure 6 Breakdown of running time for the three PD based algorithms on the Uniform synthetic dataset (SY-U) with varying n or k.

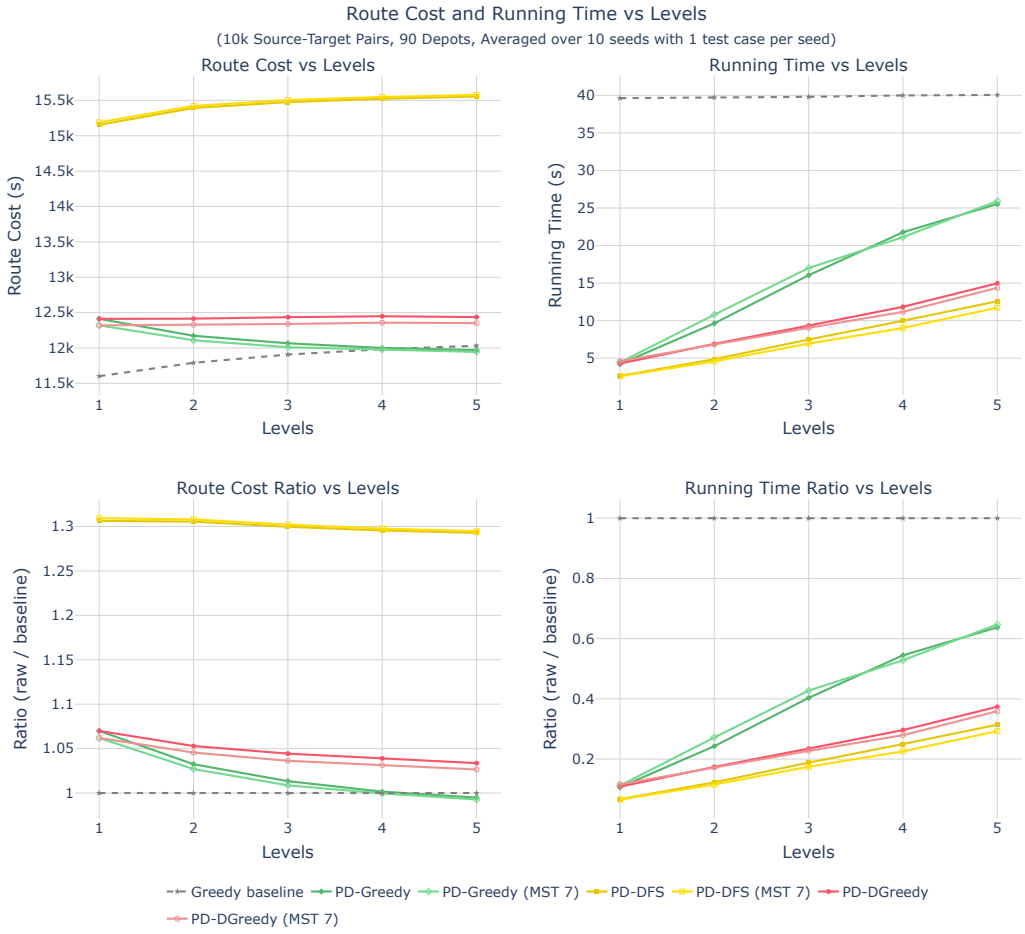


Figure 7 Results on the Uniform synthetic dataset (SY-U) with varying h.

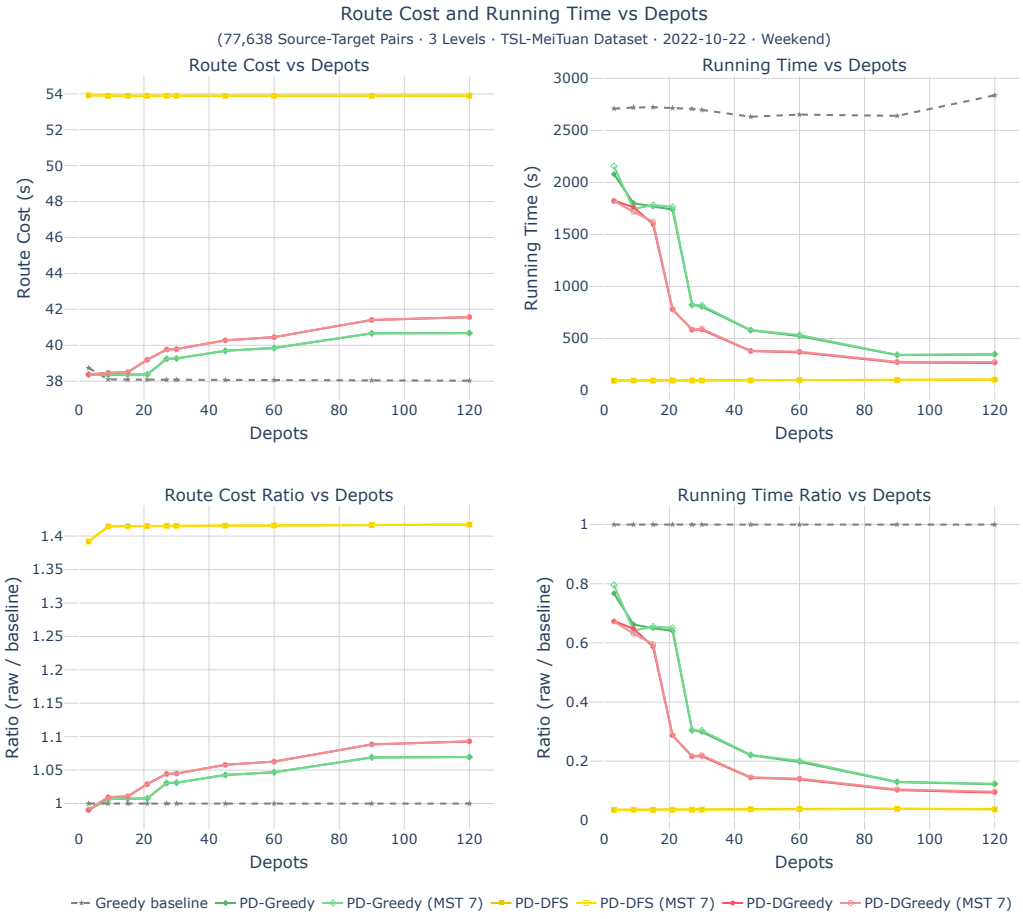


Figure 8 Results on the weekend dataset with varying numbers of depots k. For reference, when the baseline algorithm uses all depots ($k = 4901$), the total route cost is 36.12 seconds and the running time is 2731.58 seconds.

References

1 Andreas Bärtschi, Jérémie Chalopin, Shantanu Das, Yann Disser, Daniel Graf, Jan Hackfeld, and Paolo Penna. Energy-efficient delivery by heterogeneous mobile agents. In *34th Symposium on Theoretical Aspects of Computer Science (STACS 2017)*, volume 66 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.STACS.2017.10.

2 Andreas Bärtschi, Daniel Graf, and Matús Mihalák. Collective fast delivery by energy-efficient agents. In *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, volume 117 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 56:1–56:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.MFCS.2018.56.

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

4 Jérémie Chalopin, Shantanu Das, Matúš Mihal'ák, Paolo Penna, and Peter Widmayer. Data delivery by energy-constrained mobile agents. In *9th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (ALGOSENSORS 2013)*, volume 8243 of *Lecture Notes in Computer Science*, pages 111–122. Springer, 2014. doi:10.1007/978-3-642-45346-5_9.

5 Jérémie Chalopin, Riko Jacob, Matúš Mihalák, and Peter Widmayer. Data delivery by energy-constrained mobile agents on a line. In *41st International Colloquium on Automata, Languages, and Programming (ICALP 2014), Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 423–434. Springer, 2014. doi:10.1007/978-3-662-43951-7_36.

6 Jérémie Chalopin, Shantanu Das, Yann Disser, Arnaud Labourel, and Matúš Mihalák. Collaborative delivery on a fixed path with homogeneous energy-constrained agents. *Theoretical Computer Science*, 868:87–96, 2021. doi:10.1016/j.tcs.2021.04.004.

7 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: Algorithms and applications, 3rd Edition*. Springer, 2008. doi:10.1007/978-3-540-77974-2.

8 Thomas Erlebach, Kelin Luo, and Frits C.R. Spieksma. Package delivery using drones with restricted movement areas. In *33rd International Symposium on Algorithms and Computation (ISAAC 2022)*, volume 248 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 49:1–49:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ISAAC.2022.49.

9 Thomas Erlebach, Kelin Luo, and Wen Zhang. WenZhang-Vivien/Tree_construction_Primal_Dual_Routing. Software, swhId: swh:1:dir:9590a63eb16bd82996f44232019407698fc42911 (visited on 2026-07-15). URL: https://github.com/WenZhang-Vivien/Tree_construction_Primal_Dual_Routing, doi:10.4230/artifacts.27041.

10 Thomas Erlebach, Kelin Luo, and Wen Zhang. Minimizing total travel time for collaborative package delivery with heterogeneous drones, 2026. doi:10.48550/arXiv.2602.19535.

11 Greg N Frederickson, Matthew S Hecht, and Chul E Kim. Approximation algorithms for some routing problems. In *17th Annual Symposium on Foundations of Computer Science (SFCS 1976)*, pages 216–227. IEEE, 1976. doi:10.1109/SFCS.1976.6.

12 Michel X Goemans and David P Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995. doi:10.1137/S0097539793242618.

13 Meituan. Meituan INFORMS TSL research challenge. https://github.com/meituan/Meituan-INFORMS-TSL-Research-Challenge, 2024. GitHub repository, accessed 2026-02-09.

14 Sivakumar Rathinam, R Ravi, J Bae, and Kaarthik Sundar. Primal-dual 2-approximation algorithm for the monotonic multiple depot heterogeneous traveling salesman problem. In *17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020)*, volume 162 of *LIPIcs*, pages 33:1–33:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.SWAT.2020.33.

15 Martin Savelsbergh and Tom Van Woensel. 50th anniversary invited article – City logistics: Challenges and opportunities. *Transportation Science*, 50(2):579–590, 2016. doi:10.1287/TRSC.2016.0675.