Highlights

On the Fast Delivery Problem with One or Two Packages*

Iago A. Carvalho, Thomas Erlebach, Kleitos Papadopoulos

- In a graph with n vertices and m edges, the fast delivery problem with one package and k agents can be solved efficiently in $\mathcal{O}(kn \log n + km)$ time.
- The fast delivery problem with two packages is NP-hard for agents with arbitrary velocities.
- The fast delivery problem with a constant number of packages is polynomial-time solvable for agents with equal velocity.

On the Fast Delivery Problem with One or Two Packages

Iago A. Carvalho[a,b,1], Thomas Erlebach[c,*], Kleitos Papadopoulos[c]

[a]*Institute of Computing, Universidade Estadual de Campinas, Brazil*
[b]*Department of Computer Science, Universidade Federal de Minas Gerais, Brazil*
[c]*School of Informatics, University of Leicester, England, UK*

Abstract

We study two problems where k autonomous mobile agents are initially located on distinct nodes of a weighted graph with n nodes and m edges. Each agent has a predefined velocity and can only move along the edges of the graph. The first problem is to deliver one package from a source node to a destination node. The second is to simultaneously deliver two packages, each from its source node to its destination node. These deliveries are achieved by the collective effort of the agents, which can carry and exchange a package among them. For one package, we propose an $\mathcal{O}(kn \log n + km)$ time algorithm for computing a delivery schedule that minimizes the delivery time. For two packages, we show that the problem of minimizing the maximum or the sum of the delivery times is NP-hard for arbitrary agent velocities, but polynomial-time solvable for agents with equal velocity.

Keywords: Mobile agents, Dijkstra's algorithm, Polynomial-time algorithm, Time-dependent shortest paths, NP-hardness

[⋆]A preliminary version of this work has been presented at the 22nd International Symposium on Fundamentals of Computation Theory (FCT 2019) [1].

[*]Corresponding author

Email addresses: iagoac@ic.unicamp.br (Iago A. Carvalho), te17@leicester.ac.uk (Thomas Erlebach), kleitospa@gmail.com (Kleitos Papadopoulos)

URL: https://iagoac.github.io/ (Iago A. Carvalho)

[1]Iago A. Carvalho was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

1. Introduction

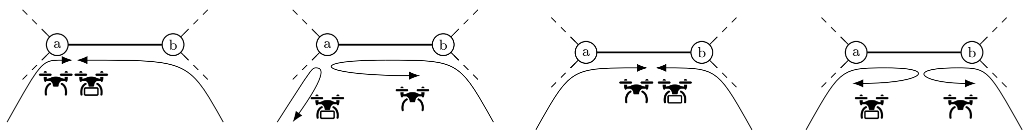
Enterprises, such as DHL, UPS, Swiss Post, and Amazon, are now delivering goods and packages to their clients using *autonomous drones* [2, 3]. Those drones depart from a base (which can be static, such as a warehouse [4], or mobile, such as a truck or a van [5]) and deliver the package into their clients' houses or in the street. However, packages are not delivered to a client that is too far from the drone's base due to the energy limitations of such autonomous aerial vehicles.

In the literature, we find some proposals for delivering packages over a longer distance. One of them, proposed by Hong, Kuby, and Murray [4], is to install recharging bases in several spots, which allows a drone to stop, recharge, and continue its path. However, this strategy may result in a delayed delivery, because drones may stop several times to recharge during a single delivery.

A manner to overcome this limitation is to use a *swarm* of drones. The idea of this technique is to position drones in recharging bases all over the delivery area. Therefore, a package can be delivered from one place to another through the collective effort of such drones, which can exchange packages among them to achieve a faster delivery. One may note that, when not carrying a package, a drone is stationed in its recharging base, waiting for the next package arrival. The problem of computing a package delivery schedule with minimum delivery time for a single package is called the FastDelivery problem [6].

We can model the input to the FastDelivery problem as a graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, with a positive length l_e associated with each edge $e \in E$, and a set of k *autonomous mobile agents* (e. g., autonomous drones) located initially on distinct nodes $p_1, p_2, \ldots, p_k$ of G. Each agent i has a predefined velocity (or speed) $\nu_i > 0$. Mobile agent i can traverse an edge e of the graph in l_e/ν_i time. The package handover between agents can be done on the nodes of the graph or in any point of the graph's edges, as exemplified in Fig. 1. The objective of FastDelivery is to deliver a single package, initially located in a source node $s \in V$, to a target node $y \in V$ while minimizing the delivery time $\mathcal{T}$.

Bärtschi et al. [6] also consider the case where each agent i is additionally associated with a weight $\omega_i > 0$ and consumes energy $\omega_i \cdot l_e$ when traversing edge e. For this model, the total energy consumption $\mathcal{E}$ of a solution becomes relevant as well, and one can consider the objective of minimizing $\mathcal{E}$ among



(a) (b)

Figure 1: (a) Package exchange on a node; (b) package exchange on an edge.

all solutions that have the minimum delivery time $\mathcal{T}$ (or vice versa), or of minimizing a convex combination $\varepsilon \cdot \mathcal{T} + (1-\varepsilon) \cdot \mathcal{E}$ for a given $\varepsilon \in (0,1)$. In this paper, we do not consider the energy consumption.

We also study a variant of FastDelivery with two packages, which is denoted by FastDelivery-2. Here, one package needs to be delivered from s_1 to y_1 and the other from s_2 to y_2, where $s_1, s_2, y_1, y_2 \in V$. It is assumed that a mobile agent cannot carry more than one package simultaneously. Let $\mathcal{T}_i$ denote the delivery time of package i, for $i \in \{1,2\}$. We consider two different objective functions: The first is the *min-max* objective function, which minimizes the maximum between $\mathcal{T}_1$ and $\mathcal{T}_2$. The second is the *min-sum* objective function, which minimizes $\mathcal{T}_1 + \mathcal{T}_2$. For the case of agents with equal speed, we also study the problem variant with an arbitrary number c of packages, denoted FastMultiDelivery.

1.1. Related Work

The problem of delivering packages through a swarm of autonomous drones has been studied in the literature. The work of Bärtschi et al. [7] considers the problem of delivering packages while minimizing the total energy consumption of the drones. In their work, all drones have the same velocity but may have different weights, and the package's exchanges between drones are restricted to take place on the graph's nodes. They show that this problem is NP-hard when an arbitrary number of packages need to be simultaneously delivered, but can be solved in polynomial time for a single package, with complexity $\mathcal{O}(k + n^3)$.

When minimizing only the delivery time $\mathcal{T}$, one can solve the problem of delivering a single package with autonomous mobile agents with different velocities in polynomial-time: Bärtschi et al. [6] gave an $\mathcal{O}(k^2 m + kn^2 + \text{APSP})$ algorithm for this problem, where APSP stands for the time complexity of the All-Pairs Shortest Paths problem in an undirected graph with n nodes and m edges. Closer inspection shows that their algorithm only requires the shortest-path distances between the k initial agent locations and all other nodes of the graph, and hence the APSP term in the running time can be replaced by $\mathcal{O}(k(m+n\log n))$ for executing Dijkstra's algorithm (implemented with Fibonacci heaps as priority queue [8]) k times, yielding a running time of $\mathcal{O}(k^2m+kn^2)$ for their algorithm for the FastDelivery problem. For the problem with many packages, Bärtschi [9, Chapter 3.2] showed NP-hardness for both the min-sum and the min-max objective, even if the graph is planar and there is a single agent (no matter whether the agent can carry only one package at a time or is able to carry multiple packages simultaneously). This shows that the problem is NP-hard for many packages also if all agents have the same speed. To the best of our knowledge, the complexity of the problem for a constant number of packages has been open.

Some work in the literature considered the minimization of both the total delivery time and the energy consumption. It was shown that the problem of delivering a single package with autonomous agents of different velocities and weights is solvable in polynomial-time when lexicographically minimizing the tuple $(\mathcal{E},\mathcal{T})$ [10]. On the other hand, it is NP-hard to lexicographically minimize the tuple $(\mathcal{T},\mathcal{E})$ or a convex combination of both parameters [6].

A closely related problem is the BudgetedDeliveryProblem (BDP) [11, 12, 13], in which a package needs to be delivered by a set of energy-constrained autonomous mobile agents. In BDP, the objective is to compute a route to deliver a single package while respecting the energy constraints of the autonomous mobile agents. This problem is weakly NP-hard in line graphs [11] and strongly NP-hard in general graphs [12]. A variant of this problem is the ReturningBudgetedDeliveryProblem (RBDP) [13], which imposes the additional constraint that the energy-constrained autonomous agents must return to their original positions after carrying the package. Surprisingly, this new restriction makes RBDP solvable in polynomial time in trees. However, it is still strongly NP-hard even for planar graphs.

Gasieniec et al. [14] studied a variant of the classical search problem, also known as the cow-path problem. In this problem variant, an agent aims to reach the location of a target as quickly as possible and the search space contains additional *expulsion points*. Visiting an expulsion point updates the speed of the agent to the maximum between its current speed and the expulsion speed associated with that expulsion point. They present online and offline algorithms for one- and two-dimensional search.

1.2. Our Contributions

For the FastDelivery problem, we provide an $\mathcal{O}(kn \log n + km)$ time algorithm for computing a delivery schedule with the minimum delivery time. This is more efficient than the previously known $\mathcal{O}(k^2m+kn^2)$ time algorithm for this problem [6]. For the FastDelivery-2 problem, we prove that it is NP-hard for both the min-sum and the min-max objective functions. While NP-hardness was known for the case with a large number of packages [9], our result shows that, surprisingly, the problem is NP-hard even for just two packages. For the special case where all agents have the same speed, we show that the problem can be solved optimally in polynomial time for any constant number of packages.

The remainder of the paper is structured as follows. Preliminaries are presented in Section 2. Then, we describe our algorithm to solve FastDelivery in Section 3. The algorithm uses as a subroutine, called once for each edge of G, an algorithm for a problem that we refer to as FastLineDelivery, which is presented in Section 4. In Section 5, we prove that FastDelivery-2 is NP-hard for both the min-max and the min-sum objective functions, and we show that the problem can be solved in polynomial time for any constant number of packages if all the agents have the same speed. Conclusions are presented in Section 6.

2. Preliminaries

As mentioned in Section 1, in the FastDelivery problem we are given an undirected graph $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges. Each edge $e \in E$ has a positive length l_e. We denote by $d(u, v)$ the sum of the lengths of the edges on a shortest path (with respect to edge lengths) from u to v in G. Generalizing the standard terminology of paths in graphs, we allow paths that can start on a node or in some point in the interior of an edge. Analogously, paths can end on a node or in some point in the interior of an edge. The length of a path is equal to the sum of the lengths of its edges. If a path starts or ends at a point in the interior of an edge, only the portion of its length that is traversed by the path is counted. For example, a path that is entirely contained in an edge $e = \{u, v\}$ of length $l_e = 10$ and starts at distance 2 from u and ends at distance 5 from u has length 3.

We are also given a number $k \leq n$ of mobile agents, which are initially located at nodes $p_1, p_2, \ldots, p_k \in V$. Each agent i has a positive velocity (or speed) ν_i, $1 \leq i \leq k$. A single package is located initially (at time 0) on a given source node $s \in V$ and needs to be delivered to a given target node $y \in V$. An agent can pick up the package in one location and drop it off (or hand it to another agent) in another one. An agent with velocity ν_i takes time d/ν_i to carry a package over a path of length d. The objective of FastDelivery is to determine a schedule for the agents to deliver the package to node y as quickly as possible, *i.e.*, to minimize the time $\mathcal{T}$ when the package reaches y.

For an instance of FastDelivery, we assume that there is at most one agent on each node. This assumption can be justified by the fact that, if there were several agents on the same node, we would use only the fastest one among them. Therefore, as already observed in [6], after a preprocessing step running in time $\mathcal{O}(k + |V|)$, we may assume that $k \leq n$.

The following lemma from [6] establishes some useful properties of an optimal delivery schedule for the mobile agents.

Lemma 1 (Bärtschi et al., 2018). *For every instance of* FastDelivery, *there is an optimum solution in which* (i) *the velocities of the involved agents are strictly increasing, and* (ii) *no involved agent arrives at its pick-up location earlier than the package (carried by the preceding agent).*

Lemma 1 implies that an agent carries a package at most once during the delivery, as the velocity of the carrying agent is monotonically increasing. This implication will be useful in the proof of Theorem 2.

In the FastDelivery-2 problem, the input is the same as for the FastDelivery problem, except that there are two packages, each specifying a source and a destination node. At any time, each agent can carry at most one of the two packages. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ denote the time when the agents deliver the first and second package, respectively, to their destinations. With the *min-max* objective, the goal it to determine a schedule that minimizes $\max\{\mathcal{T}_1, \mathcal{T}_2\}$. With the *min-sum* objective, the goal it to determine a schedule that minimizes $\mathcal{T}_1 + \mathcal{T}_2$.

3. Algorithm for the Fast Delivery Problem

Bärtschi et al. [6] present a dynamic programming algorithm that computes an optimum solution for FastDelivery in time $\mathcal{O}(k^2m + kn^2) \subseteq \mathcal{O}(k^2n^2) \subseteq \mathcal{O}(n^4)$ (where we omit the APSP term of the running time stated in [6], as discussed in Section 1.1). We design an improved algorithm, shown as Algorithm 1, with running time $\mathcal{O}(km + nk \log n) \subseteq \mathcal{O}(n^3)$ by showing that the problem can be solved by adapting the approach of Dijkstra's algorithm for edges with time-dependent transit times [15, 16]. We will prove the following theorem.

Theorem 2. *Algorithm 1 computes an optimal solution to the* FastDelivery *problem in $\mathcal{O}(nk \log n + mk)$ time.*

For any edge $\{u, v\}$, we denote by $a_t(u, v)$ the earliest time for the package to arrive at v if the package is at node u at time t and needs to be carried over the edge $\{u, v\}$. We refer to the subproblem of computing $a_t(u, v)$, for a given value of t that represents the earliest time when the package can reach u, as FastLineDelivery. Solving this problem efficiently is a crucial part of our algorithm. In Section 4, we will show that FastLineDelivery can be solved in $\mathcal{O}(k)$ time after a preprocessing step that spends $\mathcal{O}(k \log k)$ time per node. Our preprocessing calls PreprocessReceiver(v) once for each node $v \in V \setminus \{s\}$ at the start of the algorithm. Then, it calls PreprocessSender(u, t) once for each node $u \in V$, where t is the earliest time when the package can reach u. Both preprocessing steps run in $\mathcal{O}(k \log k)$ time per node. Once both preprocessing steps have been carried out, a call to FastLineDelivery(u, v, t) computes $a_t(u, v)$ in $\mathcal{O}(k)$ time.

Algorithm 1 shows the pseudo-code for our solution for FastDelivery. Initially, we run Dijkstra's algorithm to solve the single-source shortest paths problem for each node where an agent is located initially (line 2). This takes time $\mathcal{O}(k(n \log n + m))$ if we use the implementation of Dijkstra's algorithm with Fibonacci heaps as priority queue [8] and yields the distance $d(p_i, v)$ (with respect to edge lengths l_e) between any node p_i where an agent is located and any node $v \in V$. From this we compute, for every node v, the earliest time when each mobile agent can arrive at that node: The earliest possible arrival time of agent i at node v is $a_i(v) = d(p_i, v)/\nu_i$. Then, we create a list of the arrival times of the k agents on each node (line 3). For each node, we sort the list of the k agents by ascending arrival time in $\mathcal{O}(k \log k)$ time, or $\mathcal{O}(nk \log k)$ in total for all nodes. We then discard from the list of each node all agents that arrive at the same time or after an agent that is strictly faster. If several agents with the same velocity arrive at the same time, we keep one of them arbitrarily. Let $A(v)$ denote the resulting list for node v. Those lists will be used in the solution of the FastLineDelivery problem described in Section 4.

Algorithm 1: Algorithm for FastDelivery

Data: graph $G = (V, E)$ with positive edge lengths l_e and source node $s \in V$, target node $y \in V$; k agents with velocity ν_i and initial location p_i for $1 \leq i \leq k$

Result: earliest arrival time $\mathrm{dist}(y)$ for package at destination

begin
- compute $d(p_i, v)$ for $1 \leq i \leq k$ and all $v \in V$;
- construct list $A(v)$ of agents in order of increasing arrival times and velocities for each $v \in V$;
- PreprocessReceiver(v) for all $v \in V \setminus \{s\}$;
- $\mathrm{dist}(s) \leftarrow t_s$; `/* time when first agent reaches` s `*/`
- $\mathrm{dist}(v) \leftarrow \infty$ for all $v \in V \setminus \{s\}$;
- $\mathrm{final}(v) \leftarrow$ false for all $v \in V$;
- insert s into priority queue Q with priority $\mathrm{dist}(s)$;
- **while** *Q not empty* **do**
  - $u \leftarrow$ node with minimum dist value in Q;
  - delete u from Q;
  - $\mathrm{final}(u) \leftarrow$ true;
  - **if** $u = y$ **then**
    - break;
  - **end**
  - $t \leftarrow \mathrm{dist}(u)$; `/* time when package reaches` u `*/`
  - PreprocessSender(u, t);
  - **forall** *neighbors v of u with* $\mathrm{final}(v) =$ false **do**
    - $a_t(u, v) \leftarrow$ FastLineDelivery(u, v, t);
    - **if** $a_t(u, v) < \mathrm{dist}(v)$ **then**
      - $\mathrm{dist}(v) \leftarrow a_t(u, v)$;
      - **if** $v \in Q$ **then**
        - decrease priority of v to $\mathrm{dist}(v)$;
      - **else**
        - insert v into Q with priority $\mathrm{dist}(v)$;
      - **end**
    - **end**
  - **end**
- **end**
- **return** $\mathrm{dist}(y)$;

end

For each node v, we maintain a value $\text{dist}(v)$ that represents the current upper bound on the earliest time when the package can reach v (lines 5 and 6). The algorithm maintains a priority queue Q containing nodes that have a finite dist value, with the dist value as the priority (line 8). In each step, a node u with minimum dist value is removed from the priority queue (lines 10 and 11), and the node becomes *final* (line 12). Nodes that are not final are called *non-final*. The dist value of a final node will not change any more and represents the earliest time when the package can reach the node (line 16). After u has been removed from the priority queue, we compute for each non-final neighbor v of u the time $a_t(u,v)$, where $t = \text{dist}(u)$, by solving the FastLineDelivery problem (line 19). If v is already in Q, we compare $a_t(u,v)$ with $\text{dist}(v)$ and, if $a_t(u,v) < \text{dist}(v)$, update $\text{dist}(v)$ to $\text{dist}(v) = a_t(u,v)$ and adjust the priority of v in Q accordingly (line 23). On the other hand, if v is not yet in Q, we set $\text{dist}(v) = a_t(u,v)$ and insert v into Q (line 25).

Let t_s be the earliest time when an agent reaches s (or 0, if an agent is located at s initially). Let i' be that agent. As the package must stay at s from time 0 to time t_s, we can assume that i' brings the package to s at time t_s. Therefore, we initially set $\text{dist}(s) = t_s$ and insert s into the priority queue Q with priority t_s. The algorithm terminates when y becomes final (line 14) and returns the value $\text{dist}(y)$, *i.e.*, the earliest time when the package can reach y. The schedule that delivers the package to y by time $\text{dist}(y)$ can be constructed in the standard way, by storing for each node v the predecessor node u such that $\text{dist}(v) = a_{\text{dist}(u)}(u,v)$ and the schedule of the solution to FastLineDelivery$(u, v, \text{dist}(u))$. We are now ready to prove Theorem 2.

Proof (of Theorem 2). First, we note that it is easy to see that $a_t(u,v) \leq a_{t'}(u,v)$ holds for $t' \geq t$ in our setting: If the package arrives at u at time t and if we had $a_{t'}(u,v) < a_t(u,v)$ for some $t' > t$, the package could simply wait at u until time t' and then get transported to v in the same way as if it had reached u at time t'. The package would reach v at time $a_{t'}(u,v)$, contradicting the assumption that $a_{t'}(u,v) < a_t(u,v)$. Thus, the network has the FIFO property (or non-overtaking property), and it is known that the modified Dijkstra algorithm is correct for such networks [16].

Furthermore, we can observe that concatenating the solutions of FastLineDelivery (which are computed by Algorithm 4 in Section 4 and which are correct by Theorem 3 in Section 4) over the edges of the shortest path from s to y determined by Algorithm 1 indeed gives a feasible solution to FastDelivery: Assume that the package reaches u at time t while being carried by agent i and is then transported from u to v over edge $\{u, v\}$, reaching v at time $a_t(u, v)$. The only agents involved in transporting the package from u to v in the solution returned by FastLineDelivery(u, v, t) will have velocity at least ν_i because agent i arrives at u before time t, *i.e.*, $a_i(u) \leq t$, and hence no slower agent would be used to transport the package from u to v. These agents have not been involved in transporting the package from s to u by property (i) of Lemma 1, except for agent i who is indeed available at node u from time t.

The running time of the algorithm consists of the following components: Computing standard shortest paths with respect to the edge lengths l_e from the locations of the agents to all other nodes takes $\mathcal{O}(k(n \log n + m))$ time. The time complexity of the Dijkstra algorithm with time-dependent transit times for a graph with n nodes and m edges is $\mathcal{O}(n \log n + m)$. The only extra work performed by our algorithm consists of $\mathcal{O}(k \log k)$ pre-processing time for each node and $\mathcal{O}(k)$ time per edge for solving the FastLineDelivery problem, a total of $\mathcal{O}(nk \log k + mk) \subseteq \mathcal{O}(nk \log n + mk)$ time. □

4. An Algorithm for Fast Line Delivery

In this section we present the solution to FastLineDelivery that was used as a subroutine in the previous section. We consider the setting of a single edge $e = \{u, v\}$ with end nodes u and v. The objective is to deliver the package from node u to node v over edge e as quickly as possible. In our illustrations, we use the convention that v is drawn on the left and u is drawn on the right. We assume that the package reaches u at time t (where t is the earliest possible time when the package can reach u) while being carried by an agent $\bar{a}$. We will prove the following theorem.

Theorem 3. *Algorithm 4 solves* FastLineDelivery(u, v, t) *in* $\mathcal{O}(k)$ *time, assuming that* PreprocessReceiver(v) *and* PreprocessSender(u, t)*, which take time* $\mathcal{O}(k \log k)$ *each, have already been executed.*

The fastest delivery of the package over the edge from u to v where the package makes the maximum possible progress towards v at any time could in general have the following form: First, agent $\bar{a}$ will start to carry the package towards v. Then, repeatedly one of the following two types of handover events will happen: Either a faster agent coming from u will catch up with the agent currently carrying the package, take over the package, and start to carry it further towards v; or a faster agent coming from v will reach the package-carrying agent, take over the package, turn around, and start to move back towards v with the package. Solving an instance of FastLineDelivery in $\mathcal{O}(k^2)$ time would be fairly straightforward, because it is not difficult to determine the next such handover event in $\mathcal{O}(k)$ time. Our contribution is to show that FastLineDelivery can be solved in $\mathcal{O}(k)$ time provided that a preprocessing step that takes $\mathcal{O}(k \log k)$ time has been carried out for u and v beforehand. The key idea is to use a geometric representation of the agent movements and employ techniques from computational geometry to determine the handover events efficiently. In particular, the movements of the agents potentially coming from u and helping to transport the package can be represented as the lower envelope L of the corresponding line segments, and the agents potentially coming from v to help with the package delivery can be represented as a planar arrangement. It then suffices to trace L and, at each intersection point with the planar arrangement that corresponds to a meeting point with a faster agent, update L by adding a line segment corresponding to that faster agent.

As discussed in the previous section, let $A(v) = (a_1, a_2, \ldots, a_\ell)$ be the list of agents possibly arriving at node v in order of increasing velocities and increasing arrival times. For $1 \leq i \leq \ell$, denote by t_i the time when a_i reaches v, and by ν_i the velocity of agent a_i. We have $t_i < t_{i+1}$ and $\nu_i < \nu_{i+1}$ for $1 \leq i < \ell$.

Let $B(u) = (b_1, b_2, \ldots, b_r)$ be the list of agents with increasing velocities and increasing arrival times possibly arriving at node u, starting with the agent $\bar{a}$ whose arrival time is set to t. The list $B(u)$ can be computed from $A(u)$ in $\mathcal{O}(k)$ time by discarding all agents slower than $\bar{a}$ and setting the arrival time of $\bar{a}$ to t. Note that $B(u)$ cannot contain any agent that is faster than $\bar{a}$ and arrives at u before t because such an agent would have travelled towards the package and picked it up from $\bar{a}$ before time t. For $1 \leq i \leq r$, let t'_i denote the time when b_i reaches u, and let ν'_i denote the velocity of b_i. We have $t'_i < t'_{i+1}$ and $\nu'_i < \nu'_{i+1}$ for $1 \leq i < r$.

As k is the total number of agents, we have $\ell \leq k$ and $r \leq k$. In the following, we first introduce a geometric representation of the agents and their potential movements in transporting the package from u to v (Section 4.1) and then present the algorithm for FastLineDelivery (Section 4.2).

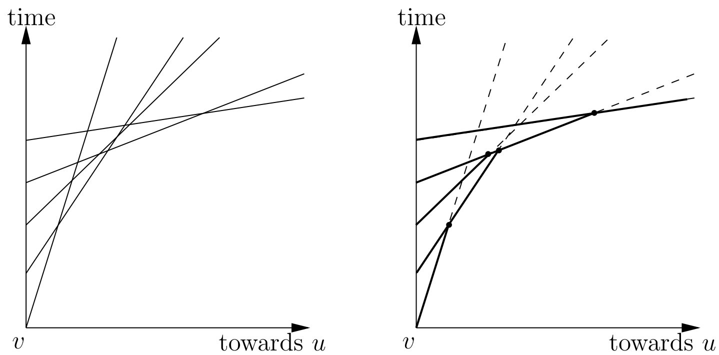


Figure 2: Geometric representation of agents moving from v towards u (left), and their relevant arrangement with removed half-lines shown dashed (right).

4.1. Geometric Representation and Preprocessing

Figure 2 shows a geometric representation of how agents $a_1, \ldots, a_\ell$ move towards u if they start to move from v to u immediately after they arrive at v. The vertical axis represents time, and the horizontal axis represents the distance from v (in the direction towards u or, more generally, any neighbor of v). The movement of each agent a_i can be represented by a line with the line equation $y = t_i + x/\nu_i$ (*i.e.*, the y value is the time when agent a_i reaches the point at distance x from v). After an agent is overtaken by a faster agent, the slower agent is no longer useful for picking up the package and returning it to v, so we can discard the part of the line of the slower agent that lies to the right of such an intersection point with the line of a faster agent. After doing this for all agents (only the fastest agent a_ℓ does not get overtaken and will not have part of its line discarded), we obtain a representation that we call the *relevant arrangement* Ψ of the agents $a_1, \ldots, a_\ell$. In the relevant arrangement, each agent a_i is represented by a line segment that starts at $(0, t_i)$, lies on the line $y = t_i + x/\nu_i$, and ends at the first intersection point between the line for a_i and the line of a faster agent a_j, $j > i$. For the fastest agent a_ℓ, there is no faster agent, and so the agent is represented by a half-line. One can view the relevant arrangement as representing the set of all points where an agent from $A(v)$ travelling towards u could receive the package from a slower agent travelling towards v.

The relevant arrangement has size $\mathcal{O}(k)$ because each intersection point can be charged to the slower of the two agents that create the intersection. It can be computed in $\mathcal{O}(k \log k)$ time using a sweep-line algorithm very similar to the algorithm by Bentley and Ottmann [17] for line segment intersection.

The relevant arrangement is created by a call to PreprocessReceiver(v) (see Algorithm 2).

Algorithm 2: Algorithm PreprocessReceiver(v)

Data: Node v (and list $A(v)$ of agents arriving at v)
Result: Relevant arrangement Ψ
1 Create a line $y = t_i + x/\nu_i$ for each agent a_i in $A(v)$;
2 Use a sweep-line algorithm (starting at $x = 0$, moving towards larger x values) to construct the relevant arrangement Ψ;

Algorithm 3: Algorithm PreprocessSender(u, t)

Data: Node u (and list $A(u)$ of agents arriving at u), time t when package arrives at u (carried by agent $\bar{a}$)
Result: Lower envelope L of agents carrying package away from u
1 $B(u) \leftarrow A(u)$ with agents slower than $\bar{a}$ removed and arrival time of $\bar{a}$ set to t;
2 Create a line $y = t'_i - x/\nu'_i$ for each agent b_i in $B(u)$;
3 Use a sweep-line algorithm (starting at $x = 0$, moving towards smaller x values) to construct the lower envelope L;

For the agents in the list $B(u) = (b_1, \ldots, b_r)$ that move from u towards v, we use a similar representation. However, in this case we only need to determine the lower envelope of the lines representing the agents. See Fig. 3 for an example. The lower envelope L has size $\mathcal{O}(k)$ and can be computed in $\mathcal{O}(k \log k)$ time[2] (e.g., using a sweep-line algorithm, or via computing the convex hull of the points that are dual to the lines [18, Sect. 11.4]). The call PreprocessSender(u, t) (see Algorithm 3) determines the list $B(u)$ from $A(u)$ and t in $\mathcal{O}(k)$ time and then computes the lower envelope of the agents in $B(u)$ in time $\mathcal{O}(k \log k)$. When we consider a particular edge $e = \{u, v\}$, we place the lower envelope L in such a way that the position on the x-axis that represents u is at $x = l_e$. We say in this case that the lower envelope

[2]Actually it would be possible to compute the lower envelope L in $\mathcal{O}(k)$ time since the lines are given to us ordered by y-intercept and slope, but since we already spend $\mathcal{O}(k \log k)$ time at each node to produce the sorted list of agent arrivals (see Step 2 of Algorithm 1 in Section 3), we do not explore such opportunities for improvement.

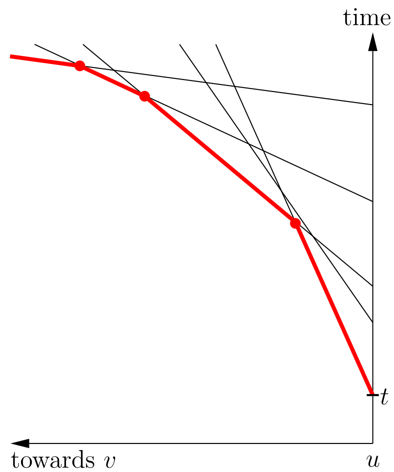


Figure 3: Geometric representation of agents moving from u towards v (lower envelope highlighted).

is *anchored* at $x = l_e$. Algorithm 3 creates the lower envelope anchored at $x = 0$, and the lower envelope anchored at $x = l_e$ can be obtained by shifting it right by l_e.

4.2. Main Algorithm for Fast Line Delivery

Assume we have computed the relevant arrangement Ψ of the agents in the list $A(v) = (a_1, \ldots, a_\ell)$ and the lower envelope L of the lines representing the agents in the list $B(u) = (b_1, b_2, \ldots, b_r)$.

The lower envelope L of the agents in $B(u)$ represents the fastest way for the package to be transported from u to v if only agents in $B(u)$ contribute to the transport and these agents move from u towards v as quickly as possible. At each time point during the transport, the package is at the closest point to v that it can reach if only agents in $B(u)$ travelling from u to v contribute to its transport. We say that such a schedule where the package is as close to v as possible at all times is *fastest and foremost* (with respect to a given set of agents).

The agents in $A(v)$ can potentially speed up the delivery of the package to v by travelling towards u, picking up the package from a slower agent that is currently carrying it, and then turning around and moving back towards v as quickly as possible. By considering intersections between L and the relevant arrangement Ψ of $A(v)$, we can find all such potential handover points. More precisely, we trace L from u (*i.e.*, $x = d(u, v)$) towards v

($i.e.$, $x = 0$). Assume that q is the first point where a handover is possible. We distinguish two cases: (1) If a faster agent j from $A(v)$ can receive the package from a slower agent i at point q of L, we update L by computing the lower envelope of L and the half-line ℓ_j representing the agent j travelling from point q towards v. This update can be implemented by tracing the lower envelope L and the half-line ℓ_j until they intersect again at a point q', and then replacing the part of L between q and q' by ℓ_j; or, if ℓ_j does not intersect L again, the part of L from q onward is replaced by ℓ_j. The time complexity for this update is $\mathcal{O}(g)$, where g is the number of line segments removed from L. (2) If the intersection point q is with an agent j from $A(v)$ that is not faster than the agent i that is currently carrying the package, we ignore the intersection point. We then continue to trace L towards v and process the next intersection point in the same way. We repeat this step until we reach v ($i.e.$, $x = 0$). The final L represents an optimum solution to the FastLineDelivery problem, and the y-value of L at $x = 0$ represents the arrival time of the package at v. See Algorithm 4 for pseudo-code of the resulting algorithm.

An illustration of step 7 of Algorithm 4, which updates L by incorporating a faster agent from $A(v)$, is shown in Fig. 4. As mentioned above, the time for executing this step is $\mathcal{O}(g)$, where g is the number of segments removed from L in the operation. As a line segment corresponding to an agent can only be removed once, the total time spent in executing step 7 (over all executions of step 7 while running Algorithm 4) is $\mathcal{O}(k)$.

Finally, we need to analyze how much time is spent in finding intersection points with line segments of the relevant arrangement Ψ while following the lower envelope L from u to v. See Fig. 5 for an illustration. We store the relevant arrangement using standard data structures for planar arrangements [19], so that we can follow the edges of each face in clockwise or counter-clockwise direction efficiently ($i.e.$, we can go from one edge to the next in constant time) and move from an edge of a face to the instance of the same edge in the adjacent face in constant time. This representation also allows us to to trace the lower envelope of Ψ in time $\mathcal{O}(k)$.

First, we remove from Ψ all line segments corresponding to agents that are not faster than $\bar{a}$ (recall that $\bar{a}$ is the agent that brings the package to node u at time t). Then, in order to find the first intersection point q_1 between L and Ψ, we can trace L and the lower envelope of Ψ from u towards v in parallel until they meet. One may observe that L cannot be above the lower envelope of Ψ at u because otherwise an agent faster than $\bar{a}$ reaches

Algorithm 4: Algorithm FastLineDelivery(u, v, t)

Data: Edge $e = \{u, v\}$, earliest arrival time t of package at u, lists $A(u)$ and $A(v)$

Result: Earliest time when package reaches v over edge $\{u, v\}$

`/*` Assume PreprocessReceiver(v) and PreprocessSender(u, t) have already been called. `*/`

$L \leftarrow$ lower envelope of agents $B(u)$ anchored at $x = l_e$;
$\Psi \leftarrow$ relevant arrangement of $A(v)$;
start tracing L from u (*i.e.*, $x = l_e$) towards v (*i.e.*, $x = 0$);
while v *(i.e.,* $x = 0$*) is not yet reached* **do**
    $q \leftarrow$ next intersection point of L and Ψ;
    `/*` assume q is intersection of agent i from L and agent j from Ψ `*/`
    if $\nu_j > \nu_i$ **then**
        replace L by the lower envelope of L and the line for agent j moving left from point q;
    else
        ignore q
    end
end
return *y-value of* L *at* $x = 0$

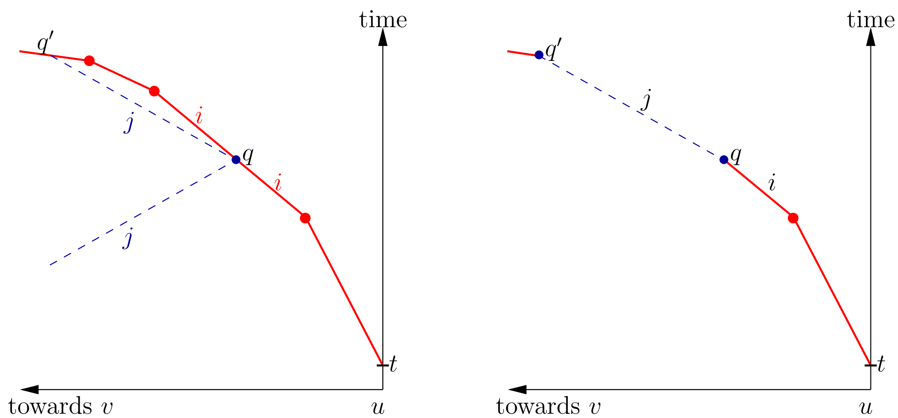


Figure 4: Agent i meets a faster agent j at intersection point q (left). The part of L from q to q' has been replaced by a line segment representing agent j carrying the package towards v (right).

u before time t, and that agent could pick up the package from $\bar{a}$ before time t and deliver it to u before time t, a contradiction to t being the earliest arrival time for the package at u. This takes $\mathcal{O}(k)$ time. After computing one intersection point q_i (and possibly updating L as shown in Fig. 4), we find the next intersection point by following the edges on the inside of the next face in counter-clockwise direction until we hit L again at q_{i+1}. This process is illustrated by the dashed arrow in Fig. 5, showing how q_2 is found starting from q_1. Hence, the total time spent in finding intersection points is bounded by the initial size of L and the number of edges of all the faces of the relevant arrangement, which is $\mathcal{O}(k)$.

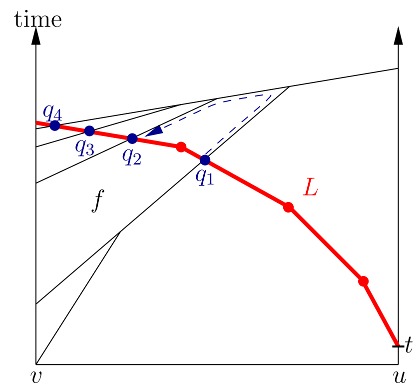


Figure 5: Intersection points q_1, q_2, q_3, q_4 between the lower envelope L (highlighted in bold) and the relevant arrangement Ψ. Point q_2 is found from q_1 by simultaneously tracing L and the edges of the face f of Ψ in counter-clockwise direction.

Proof (of Theorem 3). The claimed running time follows from the discussion above. Correctness follows by observing that the following invariant holds: If the algorithm has traced L up to position (x_0, y_0), then the current L (i.e., the result of all update operations that have been applied to L up to now) represents the fastest and foremost solution for transporting the package from u to v using only agents in $B(u)$ and agents from $A(v)$ that can reach the package by time y_0. □

5. Fast Delivery with Multiple Packages

In this section we first consider the decision version of FastDelivery-2 with min-max objective: We are given a graph $G = (V, E)$ with positive edge lengths, the source and destination node for each of the two packages, the speeds and initial locations of all agents, and a rational number H. The task is to decide if there is a schedule for the agents that delivers both packages to their respective destinations by time H. Afterwards, we consider the min-sum objective. We will prove that FastDelivery-2 is NP-hard for both the min-max and the min-sum objective functions. Finally, we consider the special case where all agents have the same speed and show that the problem, both for the min-max and the min-sum objective, can be solved optimally in polynomial time in that case for any constant number of packages. This justifies the use of agents with different velocities in the NP-hardness proof for two packages.

The remainder of this section is structured as follows. In Section 5.1, we give an overview of the ideas underlying our NP-hardness proof for the decision version of FastDelivery-2. In Section 5.2 we describe and analyze a building block that is then used as part of the reduction to show NP-hardness that is presented in Section 5.3. The special case of agents with equal speed is considered in Section 5.4.

5.1. Intuitive Overview of NP-Hardness Proof

We will prove NP-hardness of FastDelivery-2 by a reduction from the NP-complete EvenOddPartition problem [20], which is defined as follows: Given integer numbers $s_1, \ldots, s_{2n}$ with $\sum_{i=1}^{2n} s_i = 2T$, decide whether the index set $\{1, \ldots, 2n\}$ can be partitioned into two sets C and D, such that C contains either $2i-1$ or $2i$ for each i, with $\sum_{j \in C} s_j = \sum_{j \in D} s_j = T$.

A sketch of the ideas underlying the reduction is as follows. The reader may wish to look ahead at Fig. 9 on page 26 for an illustration. The graph has two separate paths P and Q of equal length, such that the first package needs to be transported along P from p to y and the second package along Q from q to z. Apart from two agents that are present at the source nodes of the two packages and carry their package the first part of the way, the majority of the delivery work is done by agents that are located at equal distance from both paths and whose speeds are increasing powers of two. For each speed 2^i, there is a pair of agents with speed 2^i, and one of them has to assist the first package and the other the second package. One of the two agents has distance $D_i - \sigma_{2i-1}$ from both paths, and the other has distance $D_i - \sigma_{2i}$ from both paths, where D_i is a suitably defined large value and σ_{2i-1} and σ_{2i} are tiny offsets that are determined by the values of s_{2i-1} and s_{2i} in the instance of EvenOddPartition. For $1 \leq i \leq n$, an agent with speed 2^i picks up the package from the agent with speed 2^{i-1} that has carried it previously (or from the initial agent), carries it for a while, and then hands it to an agent with speed 2^{i+1}.

A delivery schedule needs to choose which of the two agents with speed 2^i is used for the first package and which for the second package, and this corresponds to choosing which of the two numbers s_{2i-1} and s_{2i} is put in the set C and which in the set D of the solution to EvenOddPartition.

If the agent with distance $D_i - \sigma_{2i-1}$ carries a package, one can say that, compared to a hypothetical agent that has distance D_i from the path, this provides a "boost" of σ_{2i-1} to the package (the agent reaches the package slightly earlier, and thus makes it advance more quickly). Analogously, a boost of σ_{2i} arises if the agent with distance $D_i - \sigma_{2i}$ carries a package. The location that a package can reach by time $n+1$ then depends on all the boosts that it receives from the agents on its way. Unfortunately, the overall effect of the boosts cannot be determined by a simple addition, but requires rather lengthy and technical calculations. Nevertheless, we are able to show that a package can reach a certain point along the way to its destination (namely, the point at distance $2^{n+1} - 1 + \Delta$ from the source of the package, for a suitable value of Δ) by time $n+1$ if and only if the values of the s_i corresponding to the boosts σ_i that the package has received add up to at least T. Thus, both packages can reach that point by time $n+1$ if and only if the given instance of EvenOddPartition is a yes-instance.

Finally, an extra agent faster than all previous agents is used for each package in such a way that the agent can pick up the package at the point that has distance $2^{n+1} - 1 + \Delta$ from the package source at time $n+1$ (if the package has reached that point by that time) and deliver the package to the destination at time $n+3$. Hence, if the instance of EvenOddPartition is a yes-instance, both packages reach their destinations exactly at time $n+3$. Otherwise, at most one package can reach its destination at time $n+3$, and the other package will be delivered strictly later.

In the following sections, we present the full details of the reduction.

5.2. Building Block for One Package

Before presenting the NP-hardness proof, we discuss an important building block used in the reduction, illustrated in Fig. 6 for $n = 3$, where n is a parameter. One package needs to be delivered from p to y. There is a path from p to y, called the *horizontal path*, that consists of one edge of length 1; then two edges of length 2^{i-1} for $1 \le i \le n$ (the first such pair of edges

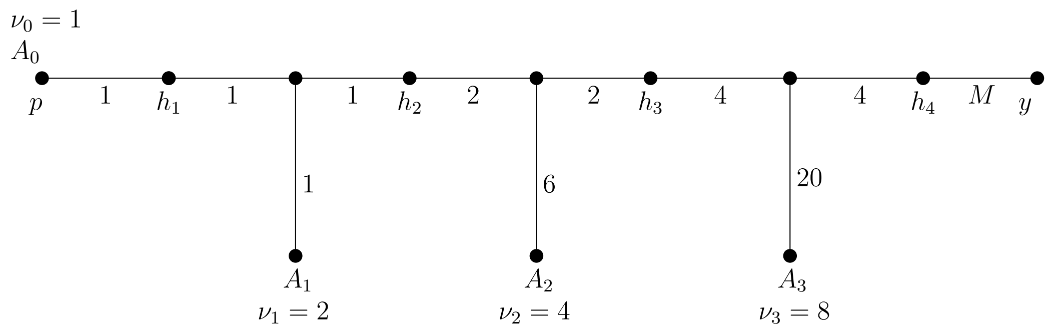


Figure 6: Building block with one package.

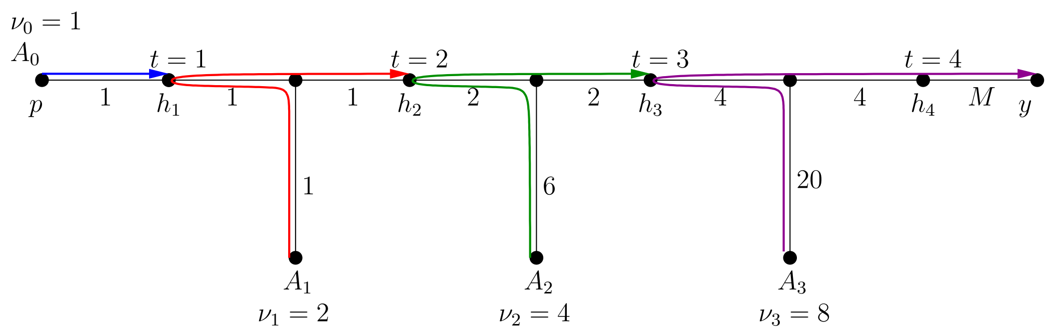


Figure 7: Optimal delivery schedule for building block.

thus also have length 1), referred to as the i-pair; and finally a single edge of length M, where the exact value of M is unimportant for the moment, it suffices to imagine it to be sufficiently large. The node at the left end of an i-pair is denoted by h_i, the node at the right end by h_{i+1}. We have agents $A_0, A_1, A_2, \ldots, A_n$ with speeds $1, 2, 4, 8, \ldots, 2^n$, respectively. Agent A_0 is initially located at p, while the other agents are initially located on vertices away from the path from p to y: The initial location of agent A_i, for $1 \leq i \leq n$, is a vertex that is connected to the middle vertex of the i-pair via an edge of length $i2^i - 2^{i-1}$.

As illustrated in Fig. 7 for $n = 3$, the optimal solution for this building block uses all the agents: Agent A_0 carries the package from p to h_1, arriving at time $t = 1$. For $1 \leq i \leq n$, A_i picks up the package at time $t = i$ at node h_i and hands it to agent A_{i+1} at time $t = i + 1$ at node h_{i+1}, or delivers it to y at time $t = n + 1 + M/2^n$ if $i = n$. In this schedule, agent A_n reaches h_{n+1} with the package at time $n + 1$.

We now consider a slightly modified instance in which the length of the edge that connects the initial location of the agent A_i to the middle vertex of the i-pair is changed from $i2^i - 2^{i-1}$ to $i2^i - 2^{i-1} - \epsilon_i$, for $1 \leq i \leq n$. Here, the ϵ_i for $1 \leq i \leq n$ are small, positive values. In particular, the values must be small enough to ensure for $1 \leq i < j \leq n$ that agent A_j cannot reach the package before agent A_i. This will speed up the delivery of the package because each agent A_i will reach the package slightly earlier than in the unmodified instance. We are interested in how far to the right of h_{n+1} the package can reach by time $n+1$ in this modified instance.

Let $\pi(x)$ denote the point on the horizontal path from p to y that has distance x from p, for any $0 \leq x \leq d(p, y)$. Note that h_i, for $i \geq 1$, corresponds to the point $\pi(2^i - 1)$.

For $i \geq 0$, let t_{i+1} be the time when agent A_{i+1} receives the package from agent A_i, and let x_{i+1} be such that $\pi(x_{i+1})$ is the point where that handover happens. A_0 picks up the package at time $t_0 = 0$ at location $p = \pi(x_0)$ with $x_0 = 0$. For $i \geq 0$, let $\lambda_i(t)$ be the function that describes the position of agent A_i in the time period from t_i to t_{i+1} (or until the agent reaches y if $i = n$; in that case, let t_{n+1} be the time when the agent reaches y), meaning that agent A_i is located at $\pi(\lambda_i(t))$ for $t_i \leq t \leq t_{i+1}$.

Lemma 4. *The following hold for all $i \geq 0$:*

$$\lambda_i(t) = 2^i t + 2^i(1-i) - 1 + \sum_{j=1}^{i} \frac{4^{i-j}\epsilon_j}{3^{i-j+1}} \tag{1}$$

$$t_{i+1} = i + 1 - \frac{1}{3 \cdot 2^i}\left(\epsilon_{i+1} + \sum_{j=1}^{i} \frac{4^{i-j}\epsilon_j}{3^{i-j+1}}\right) \tag{2}$$

$$x_{i+1} = 2^{i+1} - 1 - \frac{\epsilon_{i+1}}{3} + \frac{2}{3}\sum_{j=1}^{i} \frac{4^{i-j}\epsilon_j}{3^{i-j+1}} \tag{3}$$

Proof. We prove the lemma by induction on i. For the base case, let $i = 0$. Recall that $t_0 = 0$ and $x_0 = 0$. As agent A_0 has speed 1, we have $\lambda_0(t) = t$, which shows that (1) holds for $i = 0$. Furthermore, the original location of agent A_1 is at distance $3 - \epsilon_1$ from p and the agent travels towards p with speed 2, so the time t_1 can be calculated via

$$\lambda_0(t_1) = t_1 = 3 - \epsilon_1 - 2t_1 \Leftrightarrow t_1 = 1 - \frac{\epsilon_1}{3},$$

which shows that (2) holds for $i=0$. Furthermore, since agent A_0 travels at speed 1, we have $x_1 = t_1 = 1 - \frac{\epsilon_1}{3}$, which shows that (3) holds for $i = 0$.

For the induction step, consider any $i \geq 1$ and assume that (1)–(3) hold for $i-1$, i.e., we have:

$$\lambda_{i-1}(t) = 2^{i-1}t + 2^{i-1}(1-(i-1)) - 1 + \sum_{j=1}^{i-1} \frac{4^{(i-1)-j}\epsilon_j}{3^{(i-1)-j+1}} \tag{4}$$

$$t_i = i - \frac{1}{3\cdot 2^{i-1}}\left(\epsilon_i + \sum_{j=1}^{i-1} \frac{4^{(i-1)-j}\epsilon_j}{3^{(i-1)-j+1}}\right) \tag{5}$$

$$x_i = 2^i - 1 - \frac{\epsilon_i}{3} + \frac{2}{3}\sum_{j=1}^{i-1} \frac{4^{(i-1)-j}\epsilon_j}{3^{(i-1)-j+1}} \tag{6}$$

We will show that (1)–(3) also hold for i. As agent A_i picks up the package at time t_i at location x_i and then travels right at speed 2^i, we have for $t_i \leq t \leq t_{i+1}$:

$$\begin{aligned}
\lambda_i(t) &= x_i + (t-t_i)2^i \\
&= 2^i - 1 - \frac{\epsilon_i}{3} + \frac{2}{3}\sum_{j=1}^{i-1} \frac{4^{(i-1)-j}\epsilon_j}{3^{(i-1)-j+1}} \\
&\quad + \left(t - \left(i - \frac{1}{3\cdot 2^{i-1}}\left(\epsilon_i + \sum_{j=1}^{i-1} \frac{4^{(i-1)-j}\epsilon_j}{3^{(i-1)-j+1}}\right)\right)\right)2^i \\
&= 2^i(1-i) - 1 - \frac{\epsilon_i}{3} + \frac{2}{3}\sum_{j=1}^{i-1} \frac{4^{(i-1)-j}\epsilon_j}{3^{(i-1)-j+1}} + \frac{2}{3}\left(\epsilon_i + \sum_{j=1}^{i-1} \frac{4^{(i-1)-j}\epsilon_j}{3^{(i-1)-j+1}}\right) + t2^i \\
&= 2^i(1-i) - 1 - \frac{\epsilon_i}{3} + \frac{2\epsilon_i}{3} + \frac{4}{3}\sum_{j=1}^{i-1} \frac{4^{(i-1)-j}\epsilon_j}{3^{(i-1)-j+1}} + t2^i \\
&= 2^i(1-i) - 1 + \frac{\epsilon_i}{3} + \sum_{j=1}^{i-1} \frac{4^{i-j}\epsilon_j}{3^{i-j+1}} + t2^i \\
&= 2^i(1-i) - 1 + \sum_{j=1}^{i} \frac{4^{i-j}\epsilon_j}{3^{i-j+1}} + t2^i
\end{aligned}$$

This shows that (1) holds for i.

As the initial location of agent A_{i+1} is at distance

$$\left[2^{i+1} - 1 + 2^i\right] + \left[(i+1)2^{i+1} - 2^i - \epsilon_{i+1}\right] = (i+2)2^{i+1} - 1 - \epsilon_{i+1}$$

from p and the agent travels at speed 2^{i+1}, the time t_{i+1} when agents A_{i+1} and A_i meet can be calculated via:

$$\begin{aligned}
& \lambda_i(t_{i+1}) = (i+2)2^{i+1} - 1 - \epsilon_{i+1} - 2^{i+1}t_{i+1} \\
\Leftrightarrow \quad & 2^i t_{i+1} + 2^i(1-i) - 1 + \sum_{j=1}^{i} \frac{4^{i-j}\epsilon_j}{3^{i-j+1}} = (i+2)2^{i+1} - 1 - \epsilon_{i+1} - 2^{i+1}t_{i+1} \\
\Leftrightarrow \quad & (2^i + 2^{i+1})t_{i+1} = (3i+3)2^i - \epsilon_{i+1} - \sum_{j=1}^{i} \frac{4^{i-j}\epsilon_j}{3^{i-j+1}} \\
\Leftrightarrow \quad & t_{i+1} = \frac{(3i+3)2^i - \epsilon_{i+1} - \sum_{j=1}^{i} \frac{4^{i-j}\epsilon_j}{3^{i-j+1}}}{3 \cdot 2^i} \\
\Leftrightarrow \quad & t_{i+1} = (i+1) - \frac{1}{3 \cdot 2^i}\left(\epsilon_{i+1} + \sum_{j=1}^{i} \frac{4^{i-j}\epsilon_j}{3^{i-j+1}}\right)
\end{aligned}$$

This shows that (2) holds for i.

Finally x_{i+1} can be calculated by substituting $t = t_{i+1}$ in the expression $(i+2)2^{i+1} - 1 - \epsilon_{i+1} - 2^{i+1}t_{i+1}$ that describes the distance of A_{i+1} from p between time 0 and time t_{i+1}:

$$\begin{aligned}
x_{i+1} &= (i+2)2^{i+1} - 1 - \epsilon_{i+1} - 2^{i+1}\left((i+1) - \frac{1}{3 \cdot 2^i}\left(\epsilon_{i+1} + \sum_{j=1}^{i} \frac{4^{i-j}\epsilon_j}{3^{i-j+1}}\right)\right) \\
&= 2^{i+1} - 1 - \epsilon_{i+1} + \frac{2}{3}\left(\epsilon_{i+1} + \sum_{j=1}^{i} \frac{4^{i-j}\epsilon_j}{3^{i-j+1}}\right) \\
&= 2^{i+1} - 1 - \frac{\epsilon_{i+1}}{3} + \frac{2}{3}\sum_{j=1}^{i} \frac{4^{i-j}\epsilon_j}{3^{i-j+1}}
\end{aligned}$$

This shows that (3) also holds for i, completing the inductive step. □

Recall that the last agent that carries the package is A_n. Using $i = n$ in (1), we have that the position of agent A_n at time $t = n+1$ is equal to

$$\lambda_n(n+1) = 2^n(n+1) + 2^n(1-n) - 1 + \sum_{j=1}^{n} \frac{4^{n-j}\epsilon_j}{3^{n-j+1}}$$

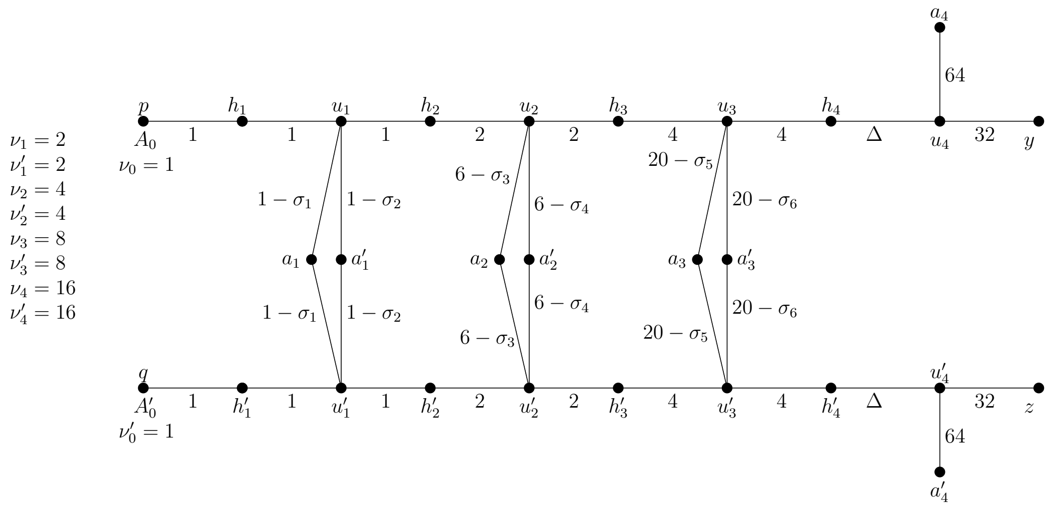


Figure 8: Illustration of reduction from EVENODDPARTITION for $n = 3$. Note that $\Delta \ll 1$.

$$= \quad 2^{n+1} - 1 + \sum_{j=1}^{n} \frac{4^{n-j}\epsilon_j}{3^{n-j+1}}\,, \tag{7}$$

This implies that the package can reach the position at distance $2^{n+1} - 1 + \Delta$ from p (for any $0 \leq \Delta \leq M$) by time $n + 1$ if and only if $\sum_{j=1}^{n} \frac{4^{n-j}\epsilon_j}{3^{n-j+1}} \geq \Delta$.

5.3. The Reduction

Theorem 5. FASTDELIVERY-2 *with min-max objective is NP-hard even in planar graphs.*

PROOF. We give a reduction from EVENODDPARTITION (defined in Section 5.1) to FASTDELIVERY-2. The EVENODDPARTITION problem is known to be (weakly) NP-complete [20].

Let an instance I of EVENODDPARTITION be given by numbers $s_1, \ldots, s_{2n}$ with $\sum_i s_i = 2T$. Without loss of generality, we can assume $s_i \leq T$ for all $1 \leq i \leq 2n$. We construct an instance I' of the fast delivery problem with two packages and $2n + 4$ agents in a graph $G = (V, E)$ as follows. See Fig. 8 for an illustration with $n = 3$.

The vertex set V of the graph G consists of $6n + 10$ vertices as follows:

$$V = \{p, q, y, z\} \cup \{h_i, h'_i, u_i, u'_i, a_i, a'_i \mid 1 \leq i \leq n + 1\}$$

There are $2n+4$ agents, denoted by $\{A_i, A'_i \mid 0 \le i \le n+1\}$. The initial location of agent A_0 is p, the initial location of agent A'_0 is q, and for $1 \le i \le n+1$, the initial location of agents A_i and A'_i are a_i and a'_i, respectively. One package must be carried from p to y, the other from q to z. The edge set E contains the following edges, where the values of the parameters σ_i, for $1 \le i \le 2n$, and Δ used to specify some of the edge lengths will be provided shortly:

- Edges $\{p, h_1\}$ and $\{q, h'_1\}$ with length 1.
- For $1 \le i \le n$:
  - Edges $\{h_i, u_i\}$, $\{u_i, h_{i+1}\}$, $\{h'_i, u'_i\}$, $\{u'_i, h'_{i+1}\}$ with length 2^{i-1}
  - Edges $\{a_i, u_i\}$ and $\{a_i, u'_i\}$ with length $i2^i - 2^{i-1} - \sigma_{2i-1}$.
  - Edges $\{a'_i, u_i\}$ and $\{a'_i, u'_i\}$ with length $i2^i - 2^{i-1} - \sigma_{2i}$.
- Edges $\{h_{n+1}, u_{n+1}\}$ and $\{h'_{n+1}, u'_{n+1}\}$ with length Δ.
- Edges $\{a_{n+1}, u_{n+1}\}$ and $\{a'_{n+1}, u'_{n+1}\}$ with length $2^{n+1}(n+1)$.
- Edges $\{u_{n+1}, y\}$ and $\{u'_{n+1}, z\}$ with length 2^{n+2}.

It is easy to see that the graph is planar. We refer to the path

$$(p, h_1, u_1, h_2, u_2, \ldots, u_n, h_{n+1}, u_{n+1}, y)$$

as P and to the path

$$(q, h'_1, u'_1, h'_2, u'_2, \ldots, u'_n, h'_{n+1}, u'_{n+1}, z)$$

as Q. We set $\Delta = 2^{-2n}$ and

$$\sigma_i = \Delta \cdot \frac{s_i}{T} \cdot \frac{3^{n+1-\lceil i/2 \rceil}}{4^{n-\lceil i/2 \rceil}}$$

for $1 \le i \le 2n$. Observe that $\sigma_i \le 3\Delta$ for all i, $1 \le i \le 2n$, since we assume $s_i \le T$. Note that all edge lengths are rational numbers whose enumerators and denominators can be specified with a number of bits that is polynomial in the size of I. Hence, the instance I' can be constructed in polynomial time.

We claim that I is a yes-instance if and only if I' admits a schedule in which both packages reach their destinations by time $n+3$.

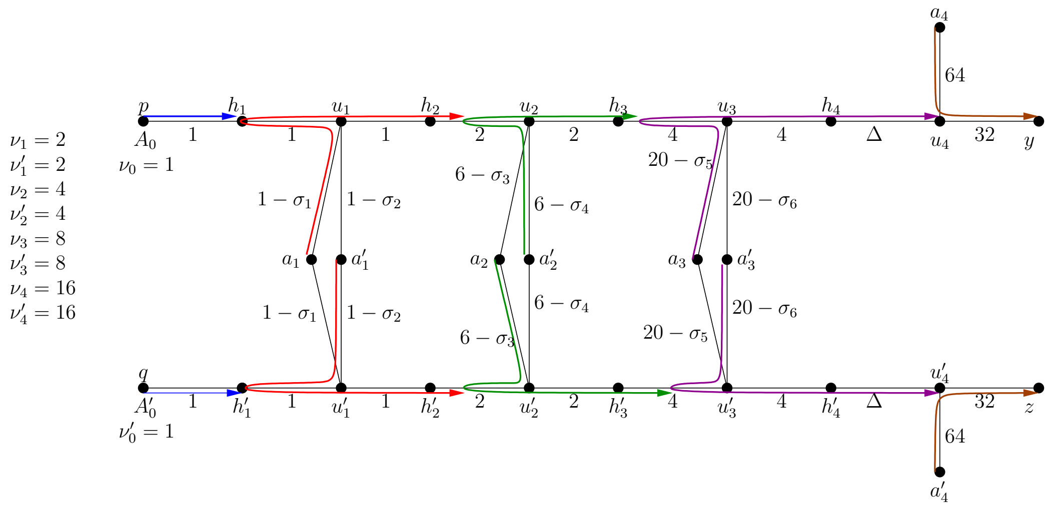


Figure 9: Illustration of delivery schedule corresponding to the solution $(\{1,4,5\},\{2,3,6\})$ of an EvenOddPartition instance.

Proof of "⇒":. Assume that I is a yes-instance. Let (C, D) be the partition of the index set $\{1, 2, \ldots, 2n\}$ such that $\sum_{j\in C} s_j = \sum_{j\in D} s_j = T$ and exactly one of $2i-1, 2i$ is in C for each $1 \leq i \leq n$. For $1 \leq i \leq n$, let $c_i = s_{2i-1}$ and $d_i = s_{2i}$ if $2i-1 \in C$, and let $c_i = s_{2i}$ and $d_i = s_{2i-1}$ otherwise. Observe that $\sum_{i=1}^n c_i = \sum_{i=1}^n d_i = T$.

For $1 \leq i \leq n$, let $Y_i = A_i$ and $Z_i = A_i'$ if $2i-1 \in C$, and $Y_i = A_i'$ and $Z_i = A_i$ otherwise. Similarly, also for $1 \leq i \leq n$, let $\epsilon_i = \sigma_{2i-1}$ and $\epsilon_i' = \sigma_{2i}$ if $2i-1 \in C$, and $\epsilon_i = \sigma_{2i}$ and $\epsilon_i' = \sigma_{2i-1}$ otherwise. Note that $\epsilon_i = \Delta \cdot \frac{c_i}{T} \cdot \frac{3^{n+1-i}}{4^{n-i}}$ and $\epsilon_i' = \Delta \cdot \frac{d_i}{T} \cdot \frac{3^{n+1-i}}{4^{n-i}}$.

We let the agents $A_0, Y_1, Y_2, \ldots, Y_n, A_{n+1}$ transport the first package from p to y along P, and the agents $A_0', Z_1, Z_2, \ldots, Z_n, A_{n+1}'$ transport the second package from q to z along Q. See Fig. 9 for an example of the resulting delivery schedule if the solution to EvenOddPartition is $(\{1,4,5\},\{2,3,6\})$. Consider the transport of the first package from p to y. Observe that the transport of the package from p to u_{n+1} by agents $A_0, Y_1, Y_2, \ldots, Y_n$ corresponds to the situation discussed in Section 5.2, and hence the findings from that section apply. By (7), at time $n+1$ the package reaches the point on P at distance

$$2^{n+1} - 1 + \sum_{j=1}^{n} \frac{4^{n-j}\epsilon_j}{3^{n-j+1}} \quad = \quad 2^{n+1} - 1 + \sum_{j=1}^{n} \frac{4^{n-j}\Delta \cdot c_j \cdot 3^{n+1-j}}{T \cdot 3^{n-j+1} 4^{n-j}}$$

$$= 2^{n+1} - 1 + \sum_{j=1}^{n} \frac{\Delta c_j}{T}$$
$$= 2^{n+1} - 1 + \Delta$$

from p. Thus, the package reaches the vertex u_{n+1} exactly at time $n+1$. Agent A_{n+1} has speed 2^{n+1} and starts at distance $2^{n+1}(n+1)$ from u_{n+1}, so it also reaches u_{n+1} at time $n+1$ and can deliver the package to y over the edge $\{u_{n+1}, y\}$ of length 2^{n+2} by time $n+3$.

The analysis of the transport of the second package from q to z is analogous: By (7), the package reaches the point on Q at distance

$$\begin{aligned} 2^{n+1} - 1 + \sum_{j=1}^{n} \frac{4^{n-j}\epsilon'_j}{3^{n-j+1}} &= 2^{n+1} - 1 + \sum_{j=1}^{n} \frac{4^{n-j}\Delta \cdot d_j \cdot 3^{n+1-j}}{T \cdot 3^{n-j+1}4^{n-j}} \\ &= 2^{n+1} - 1 + \sum_{j=1}^{n} \frac{\Delta d_j}{T} \\ &= 2^{n+1} - 1 + \Delta \end{aligned}$$

from q, i.e., the vertex u'_{n+1}, at time $n+1$. Agent A'_{n+1} reaches u'_{n+1} at the same time and can deliver the package to z by time $n+3$.

Proof of "$\Leftarrow$":. Assume there is a solution S' to I' that delivers the first package to y by time $n+3$ and the second packages to z by time $n+3$. Among all such solutions, consider one where it is not possible to decrease the delivery time of one package without increasing the delivery time of the other package. We first make some observations about the structure of the solution:

- The first package must be delivered to y at time $n+3$ by A_{n+1}, because no other agent can even reach y by time $n+3$. Furthermore, A_{n+1} must travel without ever pausing from a_{n+1} to u_{n+1} and from u_{n+1} to y, passing u_{n+1} exactly at time $n+1$. Hence, the first package must have been transported to u_{n+1} by time $n+1$ by other agents. Analogous observations hold for the second package and agent A'_{n+1}.

- The first package travels along P, and the second package travels along Q. Consider the first package. If the package were to cross over to the other path Q and then back to P, each such pair of crossings would add a length of at least $2-6\Delta+6-6\Delta = 8-12\Delta$ (a lower bound on the length of the path from u_1 to u_1' via a_1 or a_1' plus the length of the path from u_2' to u_2 via a_2 or a_2'; these are the two shortest crossings possible) to the path of that package. Furthermore, no agent can reach the package earlier on this path compared to using only path P. Hence, the detour will add a time of at least $\frac{8-12\Delta}{2^{n+1}} \geq \frac{7}{2^{n+1}}$ (as $\Delta \leq 1/16$ for $n \geq 2$, which we may assume) to the journey time of the package, and we could obtain a solution that delivers the package faster by letting it travel along P. The arguments for the second package are analogous.

- For $1 \leq i \leq n$, exactly one of the agents A_i, A_i' must be used to carry the first package, and the other to carry the second package. Assume for a contradiction that neither of the agents A_i and A_i' is used to carry the first package. As the agents A_i and A_i' have the same speed, it is clear that at most one of the two agents is used to carry the second package. Hence, one of the two agents, say, A_i, is not used at all. Then we can improve the delivery time of the first package by using A_i to take over the package from the agent A_j or A_j' with largest index $j < i$ that is used in S' to carry the package, and handing it to the agent A_j or A_j' with smallest index $j > i$ that is used in S' to carry the package. To see that A_i can indeed reach the package before A_j for any $j > i$, observe that A_i can reach p at time $(2^i - 1 + i2^i - \sigma_i)/2^i < i+1$ while A_j can reach p only at time $(2^j - 1 + j2^j - \sigma_j)/2^j = j+1-(1+\sigma_j)/2^j \geq j+0.5 \geq i+1.5$. (The argument for the second package is analogous.)

These observations imply that the findings of Section 5.2 apply to the transport of the first package on P and to the transport of the second package on Q.

For $1 \leq i \leq n$, let $\epsilon_i = \sigma_{2i-1}$ and $\epsilon_i' = \sigma_{2i}$ if A_i carries the first package, and let $\epsilon_i = \sigma_{2i}$ and $\epsilon_i' = \sigma_{2i-1}$ if A_i carries the second package. Also, let $c_i = s_{2i-1}$ and $d_i = s_{2i}$ in the former case and $c_i = s_{2i}$ and $d_i = s_{2i-1}$ in the latter case. Note that $\epsilon_i = \Delta \cdot \frac{c_i}{T} \cdot \frac{3^{n+1-i}}{4^{n-i}}$ and $\epsilon_i' = \Delta \cdot \frac{d_i}{T} \cdot \frac{3^{n+1-i}}{4^{n-i}}$.

As the first package must reach u_{n+1} and the second package must reach u_{n+1}' by time $n+1$ as shown above, we have by (7):

$$2^{n+1} - 1 + \sum_{j=1}^{n} \frac{4^{n-j}\epsilon_j}{3^{n-j+1}} \geq 2^{n+1} - 1 + \Delta$$

and

$$2^{n+1} - 1 + \sum_{j=1}^{n} \frac{4^{n-j}\epsilon'_j}{3^{n-j+1}} \geq 2^{n+1} - 1 + \Delta.$$

This means that

$$2^{n+1} - 1 + \sum_{j=1}^{n} \frac{4^{n-j}\Delta \cdot \frac{c_j}{T} \cdot \frac{3^{n+1-j}}{4^{n-j}}}{3^{n-j+1}} \geq 2^{n+1} - 1 + \Delta$$

and

$$2^{n+1} - 1 + \sum_{j=1}^{n} \frac{4^{n-j}\Delta \cdot \frac{d_j}{T} \cdot \frac{3^{n+1-j}}{4^{n-j}}}{3^{n-j+1}} \geq 2^{n+1} - 1 + \Delta.$$

Hence,

$$\sum_{j=1}^{n} \frac{c_j}{T} \geq 1$$

and

$$\sum_{j=1}^{n} \frac{d_j}{T} \geq 1.$$

As $\sum_{j=1}^{n}(c_j + d_j) = 2T$, we must have $\sum_{j=1}^{n} c_j = T$ and $\sum_{j=1}^{n} d_j = T$. Consequently, setting

$$C = \{2i-1 \mid A_i \text{ carries the first package}\} \cup \{2i \mid A'_i \text{ carries the first package}\}$$

and $D = \{1, \ldots, 2n\} \setminus C$ gives us a partition showing that I is a yes-instance of EvenOddPartition. □

Corollary 6. FastDelivery-2 *with min-sum objective is NP-hard even in planar graphs.*

Proof. The proof of Theorem 5 also gives NP-hardness for the min-sum objective: The instance constructed in the proof has the property that the sum of the delivery times is $2(n+3)$ if the instance of EvenOddPartition is a yes-instance, while the sum of the delivery times is strictly larger than $2(n+3)$ if it is a no-instance. □

Corollary 7. FastDelivery-2 *is NP-hard, for both the min-sum and the min-max objective, even if the agents can have arbitrary capacities (i.e., can carry both packages simultaneously).*

Proof. The construction in the proof of Theorem 5 is such that no advantage can be gained by having an agent carry both packages at the same time. □

Corollary 8. FastDelivery-2 *is NP-hard, for both the min-sum and the min-max objective, even if both packages have the same source and the same destination.*

Proof. We observe that the proof of Theorem 5 also works if nodes p and q are merged into one node and nodes y and z are merged into one node: As one package must reach u_{n+1} by time $n+1$ and the other must reach u'_{n+1} by time $n+1$ in any solution that delivers both packages to their joint destination by time $n+3$, it is still the case that one package must travel along P and the other along Q in any such solution. □

If we combine the assumptions of Corollaries 7 and 8, i.e., if both packages have the same source and the same destination and if the agents can carry two packages simultaneously, then the problem is polynomial-time solvable as it becomes equivalent to the FastDelivery problem.

Finally, we remark that the NP-hardness results of this section can also be used to show that the problem is NP-hard for c packages, for any constant $c > 2$: We simply add $c-2$ extra packages in a separate part of the graph, each with an agent of speed 2^{n+1} at its source node. Each of these extra packages must be delivered to a unique leaf node that is connected to the source node of the package via an edge of length $(n+3)2^{n+1}$. Thus all the extra packages can be delivered to their destinations by time $n+3$, and the agents involved in their delivery do not interact with the original instance constructed in the NP-hardness proof.

5.4. Agents with Equal Speed

Let FastMultiDelivery denote the following problem: We are given a graph $G = (V, E)$ with positive edge lengths, the source and destination node for each of $c \geq 1$ packages, and the speed ν_i and initial location p_i of agent i for $1 \leq i \leq k$. The task is to determine a delivery schedule for the agents that delivers all c packages from their sources to their respective destinations. The objective can be either the min-max objective (minimizing the time when the last package reaches its destination) or the min-sum objective (minimizing the sum of the delivery times of the c packages).

In this section we study the case where all agents have the same speed ν, i.e., $\nu_i = \nu$ for all agents i. For this case it is easy to see that it is never necessary to pass a package from one agent to another agent. If there are more than c agents placed at a node of the graph initially, we can keep c of them and discard the others because at most c agents of equal speed will be involved in delivering c packages. Therefore, we assume $k \leq cn$ from now on.

For agents with equal speed, the FastDelivery problem (with a single package) is trivial: The first agent who reaches the source s of the package carries it all the way to its destination y. For the case of an arbitrary number of packages, FastMultiDelivery is NP-hard (for both the min-max and min-sum objectives) even in the equal speed case, since the problem is NP-hard for the case of a single agent as shown by Bärtschi [9, Chapter 3.2]. We show that the problem can be solved in polynomial time for any constant number of packages. In fact, our algorithm is an FPT (fixed parameter tractable) algorithm [21] for parameter c, the number of packages, i.e., its running time is bounded by a function of the parameter times a polynomial in the size of the input.

Theorem 9. *For the case where all agents have the same speed, there is an algorithm that computes an optimal solution to* FastMultiDelivery *with min-max objective in a graph with n nodes and m edges in time $\mathcal{O}(\text{APSP} + 2^c c^{c+2.5} \cdot n^2)$, where* APSP *is the time for solving the all-pairs shortest path problem in a graph with n nodes and m edges.*

Proof. First, we consider the structure of an optimal delivery schedule. As a package will never be passed from one agent to another, each agent i that participates in the delivery of some number $j_i \geq 1$ of packages will behave as follows: It will travel to the source of the first package along a shortest path, deliver it to its destination along a shortest path, travel to the source of the second package along a shortest path, deliver it to its destination along a shortest path, and so on, until it delivers the j_i-th package. This means that once we have determined which packages an agent delivers, and in which order, then computing the best schedule for that agent is straightforward.

Denote the given packages by $K_1, K_2, \ldots, K_c$. We refer to the ordered list of packages that one agent delivers as a *package list*. For example, if an agent picks up and delivers first K_3, then K_1, and then K_5, the corresponding package list is (K_3, K_1, K_5). A solution in which g agents participate in package delivery thus induces a *partition* of the set of all packages into g package lists: All the package lists are non-empty, and each package is included in precisely one of the package lists.

Algorithm 5: Algorithm for FastMultiDelivery with equal speed

---

Data: graph $G = (V, E)$ with positive edge lengths l_e; c packages K_i with source node $s_i \in V$ and target node $y_i \in V$ for $1 \leq i \leq c$; k agents with equal velocity ν and initial location p_i for $1 \leq i \leq k$

Result: delivery schedule minimizing the maximum delivery time

begin
  forall *partitions* $\mathcal{K}$ *of* $\{K_1, \ldots, K_c\}$ *into at most* $\min\{k, c\}$ *non-empty package lists* **do**
    assume $\mathcal{K} = \{\mathcal{K}_1, \ldots, \mathcal{K}_g\}$ for some $g \leq \min\{k, c\}$;
    forall $1 \leq i \leq k, 1 \leq j \leq g$ **do**
      $T_{ij} \leftarrow$ delivery time of agent i for last package in $\mathcal{K}_j$;
    end
    construct complete bipartite graph $H = (\{1, \ldots, k\} \cup \mathcal{K}, F)$ with edge weight T_{ij} for each edge $\{i, \mathcal{K}_j\}$;
    compute a bottleneck matching $\mathcal{M}_{\mathcal{K}}$ in H;
    $\mathcal{T}_{\mathcal{K}} \leftarrow$ largest edge weight in $\mathcal{M}_{\mathcal{K}}$;
  end
  return *delivery schedule given by* $\mathcal{M}_{\mathcal{K}}$ *with minimum* $\mathcal{T}_{\mathcal{K}}$*;*
end

The idea of Algorithm 5 is now to enumerate all possible partitions $\mathcal{K}$ of the set of c packages into at most $\min\{k, c\}$ ordered package lists, to compute a delivery schedule with minimum delivery time for each such partition via a bottleneck matching algorithm, and in the end to output the best schedule found.

Let $\mathcal{K} = \{\mathcal{K}_1, \ldots, \mathcal{K}_g\}$ be a partition of the set of packages into package lists, with $1 \leq g \leq \min\{k, c\}$. Let T_{ij} be the time when agent i delivers the last package in $\mathcal{K}_j$ if agent i delivers the packages in $\mathcal{K}_j$ (and no other packages) in the given order. The total travel distance S_{ij} of agent i for delivering the packages in $\mathcal{K}_j$ can be computed by adding up the shortest-path distances from p_i to the source of the first package in $\mathcal{K}_j$, from there to the destination of that package, from there to the source of the second package in $\mathcal{K}_j$, and so on, ending with the shortest path from the source of the last package in $\mathcal{K}_j$ to its destination. The value T_{ij} can then be calculated as S_{ij}/ν.

The algorithm then builds a complete bipartite graph H with vertex sets $\{1, \ldots, k\}$ (representing agents) and $\{\mathcal{K}_1, \ldots, \mathcal{K}_g\}$ (representing package lists), where edge $\{i, \mathcal{K}_j\}$ is given weight T_{ij}. It then computes a bottleneck matching (*i.e.*, a maximum cardinality matching that minimises the largest weight of any of its edges) in H. That matching $\mathcal{M}_\mathcal{K}$, with largest edge weight $\mathcal{T}_\mathcal{K} = \max_{\{i,\mathcal{K}_j\}\in\mathcal{M}_\mathcal{K}} T_{ij}$, then corresponds to a delivery schedule with maximum delivery time $\mathcal{T}_\mathcal{K}$: For every edge $\{i, \mathcal{K}_j\}$ in the matching, agent i delivers the packages in $\mathcal{K}_j$ by time T_{ij}.

After doing this for all partitions $\mathcal{K}$, the algorithm outputs the delivery schedule corresponding to the matching $\mathcal{M}_\mathcal{K}$ for which $\mathcal{T}_\mathcal{K}$ is minimized.

It is clear that the algorithm outputs a valid delivery schedule. To see that it outputs an optimal schedule, note that in one of the iterations the algorithm will consider a partition into package lists that is the same as the one used in an optimal schedule, and the solution to the bottleneck matching problem for the resulting matching instance must then correspond to an optimal schedule (because the optimal schedule can also be interpreted as a matching between agents and package lists, and its objective value corresponds to the largest edge weight in that matching).

It remains to analyze the running time of the algorithm. The number of partitions of the set of c packages into package lists can be bounded by $c! \cdot 2^c \leq (2c)^c$, because these partitions can be generated (with duplicates) by enumerating all $c!$ permutations of the c packages and, for each of the c packages, determining whether it is the last package of its list or not (2^c possibilities).

The graph H has at most $k + c$ nodes and kc edges. If we solve the all-pairs shortest path problem in G once in the beginning in APSP $\in \mathcal{O}(n^3)$ time [22], we can determine all the edge weights of H in $\mathcal{O}(kc)$ time: For each agent i, computing the weight of the edge to vertex $\mathcal{K}_j$ requires adding up $\mathcal{O}(|\mathcal{K}_j|)$ shortest-path distances, so all weights of edges incident with agent i can be computed in $\mathcal{O}(\sum_{j=1}^{g} |\mathcal{K}_j|) = \mathcal{O}(c)$ time.

The algorithm by Punnen and Nair [23] solves the bottleneck matching problem in a bipartite graph with n' nodes and m' edges in $\mathcal{O}(n'\sqrt{n'm'})$ time, so it runs in $\mathcal{O}((k+c)\sqrt{(k+c)kc}) = \mathcal{O}(n^2c^{2.5})$ time on the graph H (recall that we can assume $k \leq nc$).

Thus the algorithm runs in time $\mathcal{O}(\text{APSP} + (2c)^c \cdot n^2c^{2.5}) = \mathcal{O}(\text{APSP} + 2^c c^{c+2.5} \cdot n^2)$. □

We remark that Theorem 9 implies a polynomial-time algorithm for FastMultiDelivery if the agents have equal speed and c is a fixed constant. Furthermore, the algorithm of Theorem 9 is an FPT algorithm [21] for the fast delivery problem with an arbitrary number of packages and agents of equal speed with respect to the number of packages as parameter.

Theorem 10. *For the case where all agents have the same speed, there is an algorithm that computes an optimal solution to* FastMultiDelivery *with min-sum objective in a graph with n nodes in time* $\mathcal{O}(2^c c^{c+6} \cdot n^3)$.

Proof. We again use Algorithm 5, but change steps 5, 8 and 9 as follows: In step 5, we set T_{ij} to the sum of the delivery times of the packages in $\mathcal{K}_j$ when agent i delivers them in the given order. In step 8, we compute a maximum cardinality matching of minimum total edge weight, instead of a bottleneck matching. In step 9, we set $\mathcal{T}_{\mathcal{K}}$ to the sum of the weights of all edges in the matching computed in step 8.

It is easy to see that the total weight of a matching equals the sum of the delivery times of all packages in the corresponding schedule, so the algorithm produces an optimal schedule.

Using the Hungarian method [24], a maximum cardinality matching of minimum total edge weight in the graph H with $\mathcal{O}(kc)$ nodes can be computed in $\mathcal{O}((kc)^3) = \mathcal{O}(n^3c^6)$ time. The overall running time is then $\mathcal{O}(\text{APSP} + (2c)^c \cdot n^3c^6)$. Since APSP is bounded by $\mathcal{O}(n^3)$ [22], the term APSP is dominated by the other term and can be omitted. □

6. Conclusion

We have presented an algorithm with improved running time $\mathcal{O}(km + nk \log n)$ for FastDelivery. The algorithm was obtained by adapting the approach of Dijkstra's algorithm for edges with time-dependent transit times. The subproblem corresponding to relaxing an edge was solved by applying techniques from computational geometry to a geometric representation of the agent movements.

Furthermore, we have shown that when a second package is added, the resulting FastDelivery-2 problem is NP-hard for both the min-max and the min-sum objective functions, even in planar graphs and even if both packages have the same source and the same destination. Previously, NP-hardness was only known for the case where the number of packages is part of the input [9]. It is worth noting that it is not clear whether the problem with multiple packages is contained in NP, since there is no obvious bound on the length of the description of the schedule that specifies the agent movements in the solution (see [9, Chapter 3.1] for further discussion of this issue). For the special case of agents with equal speed, we showed that the FastMultiDelivery problem can be solved optimally in polynomial time for any constant number of packages, for both the min-max and the min-sum objective.

An interesting direction for future work could be studying the Euclidean version of FastDelivery, where the source and destination of the package, as well as the initial locations of the agents, are points in the Euclidean plane, and the agents can move along arbitrary curves (it is clear that polylines suffice) in the plane. Future work may also study the question whether the FastMultiDelivery problem is still polynomial-time solvable for a constant number of packages and agents with equal speed when the agents can have capacities larger than 1.

References

[1] I. A. Carvalho, T. Erlebach, K. Papadopoulos, An efficient algorithm for the fast delivery problem, in: L. A. Gasieniec, J. Jansson, C. Levcopoulos (Eds.), 22nd International Symposium on Fundamentals of Computation Theory (FCT 2019), Vol. 11651 of Lecture Notes in Computer Science, Springer, 2019, pp. 171–184. doi:10.1007/978-3-030-25027-0_12.

[2] D. Bamburry, Drones: Designed for product delivery, Design Management Review 26 (1) (2015) 40–48. doi:10.1111/drev.10313.

[3] A. Regev, Drone deliveries are no longer pie in the sky (Apr 2018). URL https://www.forbes.com/sites/startupnationcentral/2018/04/10/drone-deliveries-are-no-longer-pie-in-the-sky/

[4] I. Hong, M. Kuby, A. Murray, A deviation flow refueling location model for continuous space: A commercial drone delivery system for urban areas, in: Advances in Geocomputation, Springer, 2017, pp. 125–132. doi:10.1007/978-3-319-22786-3_12.

[5] C. C. Murray, A. G. Chu, The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery, Transportation Research Part C: Emerging Technologies 54 (2015) 86–109. doi:10.1016/j.trc.2015.03.005.

[6] A. Bärtschi, D. Graf, M. Mihalák, Collective fast delivery by energy-efficient agents, in: I. Potapov, P. Spirakis, J. Worrell (Eds.), 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018), Vol. 117 of LIPIcs, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018, pp. 56:1–56:16. doi:10.4230/LIPIcs.MFCS.2018.56.

[7] A. Bärtschi, J. Chalopin, S. Das, Y. Disser, D. Graf, J. Hackfeld, P. Penna, Energy-efficient delivery by heterogeneous mobile agents, in: 34th Symposium on Theoretical Aspects of Computer Science (STACS 2017), Vol. 66 of LIPIcs, Schloss Dagstuhl, Leibniz-Zentrum für Informatik, 2017, p. 10. doi:10.4230/LIPIcs.STACS.2017.10.

[8] M. L. Fredman, R. E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, J. ACM 34 (3) (1987) 596–615. doi:10.1145/28869.28874.

[9] A. Bärtschi, Efficient delivery with mobile agents, Ph.D. thesis, ETH Zürich (2017).

[10] A. Bärtschi, T. Tschager, Energy-efficient fast delivery by mobile agents, in: International Symposium on Fundamentals of Computation Theory (FCT 2017), Vol. 10472 of Lecture Notes in Computer Science, Springer, 2017, pp. 82–95. doi:10.1007/978-3-662-55751-8_8.

[11] J. Chalopin, R. Jacob, M. Mihalák, P. Widmayer, Data delivery by energy-constrained mobile agents on a line, in: 41st International Colloquium on Automata, Languages, and Programming (ICALP 2014), Vol. 8573 of Lecture Notes in Computer Science, Springer, 2014, pp. 423–434. doi:10.1007/978-3-662-43951-7_36.

[12] J. Chalopin, S. Das, M. Mihalák, P. Penna, P. Widmayer, Data delivery by energy-constrained mobile agents, in: International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (ALGOSENSORS 2013), Vol. 8243 of Lecture Notes in Computer Science, Springer, 2013, pp. 111–122. doi:10.1007/978-3-642-45346-5_9.

[13] A. Bärtschi, J. Chalopin, S. Das, Y. Disser, B. Geissmann, D. Graf, A. Labourel, M. Mihalák, Collaborative delivery with energy-constrained mobile robots, Theoretical Computer Science 810 (2017) 2–14. doi:10.1016/j.tcs.2017.04.018.

[14] L. Gasieniec, S. Kijima, J. Min, Searching with increasing speeds, in: Proceedings of the 20th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2018), Vol. 11201 of Lecture Notes in Computer Science, Springer, 2018, pp. 126–138. doi:10.1007/978-3-030-03232-6_9.

[15] K. Cooke, E. Halsey, The shortest route through a network with time-dependent internodal transit times, Journal of Mathematical Analysis and Applications 14 (3) (1966) 493–498. doi:10.1016/0022-247X(66)90009-6.

[16] D. Delling, D. Wagner, Time-dependent route planning, in: R. K. Ahuja, R. H. Möhring, C. D. Zaroliagis (Eds.), Robust and Online Large-Scale Optimization: Models and Techniques for Transportation Systems, Vol. 5868 of Lecture Notes in Computer Science, Springer, 2009, pp. 207–230. doi:10.1007/978-3-642-05465-5_8.

[17] J. L. Bentley, T. Ottmann, Algorithms for reporting and counting geometric intersections, IEEE Trans. Computers 28 (9) (1979) 643–647. doi:10.1109/TC.1979.1675432.

[18] M. de Berg, O. Cheong, M. J. van Kreveld, M. H. Overmars, Computational geometry: Algorithms and applications, 3rd Edition, Springer, 2008.

[19] M. Goodrich, K. Ramaiyer, Geometric data structures, in: J.-R. Sack, J. Urrutia (Eds.), Handbook of Computational Geometry, Elsevier Science, 2000, pp. 463–489.

[20] M. R. Garey, D. S. Johnson, Computers and Intractability. A Guide to the Theory of NP-Completeness, W. H. Freeman and Company, New York-San Francisco, 1979.

[21] R. G. Downey, M. R. Fellows, Parameterized Complexity, Monographs in Computer Science, Springer, 1999. doi:10.1007/978-1-4612-0515-9.

[22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, 3rd Edition, MIT Press, 2009.
URL http://mitpress.mit.edu/books/introduction-algorithms

[23] A. P. Punnen, K. Nair, Improved complexity bound for the maximum cardinality bottleneck bipartite matching problem, Discret. Appl. Math. 55 (1) (1994) 91–93. doi:10.1016/0166-218X(94)90039-6.

[24] J. Edmonds, R. M. Karp, Theoretical improvements in algorithmic efficiency for network flow problems, J. ACM 19 (2) (1972) 248–264. doi:10.1145/321694.321699.