The Piggyback Transportation Problem: Transporting drones launched from a flying warehouse[⋆]

Kai Wang[a], Erwin Pesch[a,b], Dominik Kress[c,*], Ilia Fridman[a], Nils Boysen[d]

[a] *HHL Leipzig, Center for Advanced Studies in Management, Jahnallee 59, 04109 Leipzig, Germany*
[b] *University of Siegen, Management Information Science, Kohlbettstraße 15, 57068 Siegen, Germany*
[c] *Helmut Schmidt University - University of the Federal Armed Forces Hamburg, Business Administration, especially Procurement and Production, Friedrich-Ebert-Damm 245, 22159 Hamburg, Germany*
[d] *Friedrich-Schiller-Universität Jena, Operations Management, Carl-Zeiß-Straße 3, 07743 Jena, Germany*

Abstract

This paper treats the Piggyback Transportation Problem: A large vehicle moves successive batches of small vehicles from a depot to a single launching point. Here, the small vehicles depart toward assigned customers, supply shipments, and return to the depot. Once the large vehicle has returned and another batch of small vehicles has been loaded at the depot, the process repeats until all customers are serviced. With autonomous driving on the verge of practical application, this general setting occurs whenever small autonomous delivery vehicles with limited operating range, e.g., unmanned aerial vehicles (drones) or delivery robots, need to be brought in the proximity of the customers by a larger vehicle, e.g., a truck. We aim at the most elementary decision problem in this context, which is inspired by Amazon's novel last-mile concept, the flying warehouse. According to this concept, drones are launched from a flying warehouse and – after their return to an earthbound depot – are resupplied to the flying warehouse by an air shuttle. We formulate the Piggyback Transportation Problem, investigate its computational complexity, and derive suited solution procedures. From a theoretical perspective, we prove different important structural problem properties. From a practical point of view, we explore the impact of the two main cost drivers, the capacity of the large vehicle and the fleet size of small vehicles, on service quality.

Keywords: Scheduling, Logistics, Work sharing, Approximation algorithm

1. Introduction

There exist quite a few settings, where a larger vehicle piggybacks multiple smaller vehicles and transports them toward an advantageous launching point in order to achieve efficient transportation processes. In this context, we treat the Piggyback Transportation Problem (PTP), which can be characterized as follows (see Figure 1): There is a single large vehicle and a fleet of small vehicles. The large vehicle has a given capacity for small vehicles and moves back and forth between a depot and a launching point. At the depot, the large vehicle loads a batch of small vehicles, each of which has been loaded with a shipment for a specific customer while being in the depot, and transports them to an advantageous launching point, which is hard to reach for the small vehicles without assistance but provides easier access to their target area. At the launching point, each small vehicle departs toward its assigned customer, delivers the shipment and returns to the depot. In parallel, the large vehicle drives back to the depot, where another batch of small vehicles is loaded. This process repeats

*Corresponding author
Email addresses: kai.wang@my.cityu.edu.hk (Kai Wang), erwin.pesch@uni-siegen.de (Erwin Pesch), dominik.kress@hsu-hh.de (Dominik Kress), ilia.fridman1@gmail.com (Ilia Fridman), nils.boysen@uni-jena.de (Nils Boysen)

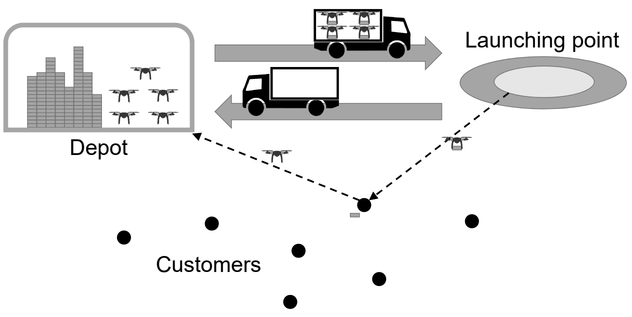


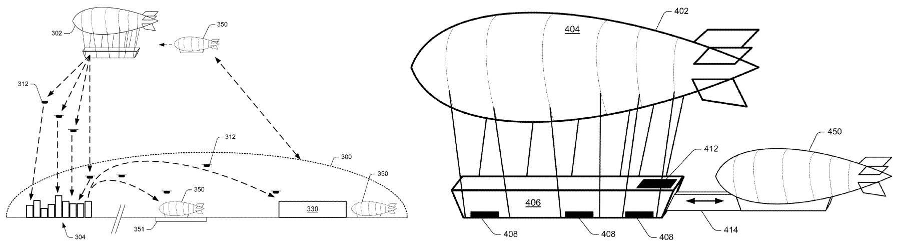
Figure 1: Schematic setting of PTP

until all shipments have been delivered to the customers. PTP decides on the assignment of small vehicles to multiple successive batches and the departure times of the large vehicle. Furthermore, each small vehicle has to be assigned to a customer for each of the batches in which it is included. Depending on the customer's location, this influences the travel time of the small vehicle back to the depot. In this setting, we aim to minimize the makespan, i.e., the time instant at which all customers have been served and the large vehicle as well as all small vehicles have returned to the depot.

1.1. Applications and Literature Review

PTP is a generic problem that has quite a few applications in the transportation or production context. With autonomous driving on the verge of practical application, this setting occurs, for instance, whenever small autonomous delivery vehicles, e.g., unmanned aerial vehicles (drones) (Otto et al. 2018) or delivery robots (Boysen et al. 2018), with limited operating range need to be brought in the proximity of customers by a larger vehicle, e.g., a truck. The customers can be of different nature. In the transportation context, they may be recipients of shipments of an online retailer. In a production context, they may be workers at an assembly line that are supplied with the needed material.

To derive basic insights, we aim at a core scheduling problem where the routing of the large vehicle is reduced to the most elementary setting, a movement back and forth between a depot and a single launching point. Such a reduced setting is, for instance, relevant for the airborne fulfillment center concept recently patented by online retailer Amazon (Berg et al. 2016). This innovative last-mile delivery concept aims to provide high-speed e-commerce deliveries to customers in urban areas from a flying warehouse, i.e., an airship hovering over a city center. Its basic elements are depicted in Figure 2a. Once a customer places an order, a drone is loaded with the



(a) Complete delivery chain (b) Replenishment process

Figure 2: Concept of an airborne fulfillment center from Amazon patent (Berg et al. 2016)

ordered goods, flies down from the airship, and delivers the parcel. When the delivery is completed, the drone does not directly return to the flying warehouse, but flies back to an earthbound depot. An air shuttle resupplies the airship with drones and inventory from the depot (see Figure 2b). Certainly, this is a concept for the farther future with many basic components still to be developed to a market-ready state. Compared to other application scenarios of drone-based home delivery, however, it has a few advantages. Most important, the overall process is very energy efficient, because, when departing from a high altitude, the drones utilize gravitation and solely have to stabilize their flights. Furthermore, the ground level flight legs are empty flights without payload. With respect to energy consumption, these flight legs are usually more costly than the flights from the airship to the customers but, nevertheless, require less operating energy than their alternatives, i.e., trips from the customers to the flying warehouse. Thus, in spite of the energy capacity of drones being notoriously scarce (Agatz et al. 2018), a large urban area can be supplied. Other concepts that aim to launch drones either directly from (earthbound) distribution centers or from trucks acting as mobile launching platforms, require a dense and costly depot network or have to apply plenty of trucks driving through the congested street networks, respectively.

In the above application of airborne fulfillment centers, the large vehicle – as defined in the context of PTP – corresponds to the air shuttle that replenishes the flying warehouse (see Figure 2b), while the small vehicles correspond to the drones. The shuttle flies back and forth between the earthbound depot and the launching point at the airship. Thus, one has to decide on the starting times of the successive shuttle trips, the drones to be taken on board, and the customer to be serviced by each drone launched from the airship. While plenty of scientific papers on launching drones from trucks have recently been written (see Otto et al. 2018), airborne warehouses have received less attention. To the best of the authors' knowledge, there is only one paper by Poikonen & Golden (2020). It is, however, concerned with the routing of the airborne warehouse and drone releases toward customers in the sky. We complement this research and are the first to schedule the resupply of the flying warehouse with drones by a shuttle from an earthbound depot. Other than Poikonen & Golden (2020), we presuppose a fixed position of the airborne warehouse. At first sight, this may seem to be a very limiting assumption that restricts the applicability of PTP to cases where the flying warehouse remains at a single spot. Note, however, that the flying warehouse concept presupposes an airship, which is not very fast and can thus only slowly adapt its position over the day. Our PTP, instead, is an operational problem, which is solved repeatedly during the day on rolling planning horizons, e.g., to adapt the drone resupply schedule to new customers and urgent orders that have newly arrived. For each single planning run, the variations in the positions of the flying warehouse are rather small, so that, at least for a slow airship, the assumption of a fixed launching point is a suitable simplification to ease schedule computations in each short-term planning run.

In this paper, we will stick to the application example of airborne warehouses, so that the small and large vehicles within PTP will also be called drones and shuttle, respectively. We furthermore note that the term piggyback transportation is sometimes reserved for the transport of (truck) trailers and semi-trailers on flat railcars (see, e.g., Thoung 1989). We, however, interpret the term in a rather broad sense and allow any kind of vehicle being given piggyback, so that there are quite a few alternative applications for our basic PTP, some of which are briefly elaborated in the following.

The small vehicles could also represent a team of car relocators moving export cars onto roll-on roll-off vessels. Large deep-sea car carriers have a capacity of up to 6000 vehicles (Cordeau et al. 2011), so that driving the cars, parked somewhere in the huge terminal area, on board of such a large vessel causes considerable car relocation effort. To speed up the process, each team of relocators is supported by a taxi bus (large vehicle) that transports them from the ship back to the field of the parking area that is currently processed (launching point) (Mattfeld & Kopfer 2003). From there, the relocators walk to their respective cars (customers) and drive them onto the vessel. Then, another taxi trip is executed starting at the vessel (depot) and so on, until all cars are loaded. Existing research on operating vehicle transshipment terminals mainly addresses the questions of where to park cars in the terminal area (Mattfeld & Orth 2006; Cordeau et al. 2011), how to coordinate the decisions in a hierarchical planning approach (Mattfeld & Kopfer 2003), and how to determine team structures (Mattfeld & Branke 2005). The operational scheduling of car movements corresponding to PTP has not been considered yet.

From a structural point of view, PTP has some similarities with job scheduling on a batching machine. A batching machine can process up to k jobs in parallel and the processing time of a batch corresponds to the maximum processing time among all parallel jobs. Corresponding elementary complexity results for the traditional machine scheduling objectives are summarized in Brucker et al. (1998) and Blazewicz et al. (2019). When minimizing the makespan, which corresponds to the objective of our PTP, batch scheduling is solvable in polynomial time (see Brucker et al. 1998). For most other objectives, the problem turns out NP-hard. Important algorithmic contributions for these settings are, for instance, provided by Wang & Uzsoy (2002), Malapert et al. (2012), as well as Jia & Leung (2015). In the context of PTP, the travel of a small vehicle from the launching point toward its designated customer and back to the depot can be interpreted as a job. The main difference of scheduling a batching machine compared to our problem, however, is that the starting time of the next batch (the next trip of the large vehicle) in the context of PTP is not determined by the maximum processing time of the previous batch. Once the large vehicle has returned to the depot, it does not have to wait for the small vehicles of the previous batch to return. It can depart instantaneously with the small vehicles that have already returned (even if they were launched in a batch prior to the last).

Another related problem setting dedicated to the transportation of different raw coal lots stored in a coal yard (depot) to different neighboring coke ovens (launching point) by a feeding train (large vehicle) is investigated by Liu et al. (2019). They, however, assume that coke ovens influence the cooling process of neighboring ovens, so that there are complex interdependencies between processing times. Our processing times, instead, only depend on different customer locations. Thus, the problems are not directly transferable and PTP requires a separate analysis and new algorithmic approaches.

1.2. Contribution and Paper Structure

This paper is the first to formulate PTP, a generic problem that has various potential applications by itself (see Section 1.1), but may also occur as a subproblem in more complex transportation settings, e.g., with multiple alternative launching points or a single launching point with a dynamically varying position. For these more involved problem settings, PTP may be a valid core problem to be solved repeatedly as a workhorse in a decomposition procedure (e.g., a metaheuristic) or as a procedure for deriving quick lower bounds.

Our further contributions are in two areas. From a theoretical point of view, we derive important problem properties:

- We prove strong NP-hardness of PTP even for very restricted special cases with solely two small vehicles.
- For the special case with identical travel times of the small vehicles for all customers, we propose a dynamic programming approach with polynomial running time.
- For the case where the capacity of the large vehicle is large enough to take the complete fleet of small vehicles on board, we propose a greedy algorithm with an approximation ratio of 2.
- We furthermore propose a polynomial-time approximation scheme (PTAS) for problem instances that are characterized by "small" travel times of the small vehicles for all customers.

- Finally, we make use of the above results to derive a polynomial algorithm with approximation ratio $(1+\epsilon)(3+\epsilon)$ for an arbitrary constant $\epsilon > 0$ for the general case of PTP.

Beyond these theoretical contributions, we also propose an integer program and a simple, yet very efficient heuristic approach for PTP. Based on a computational study, this allows to derive decision support for transportation managers that have to set up a piggyback transportation process. In most applications, the main cost drivers will certainly be the capacity of the large vehicle and the fleet size of small vehicles. We explore how to best allocate investments among these two cost drivers.

The remainder of the paper is structured as follows. In Section 2, we introduce the notation used throughout this article and define PTP in detail. A corresponding Integer Program (IP) is presented in Section 3. As outlined above, Sections 4 and 5 are concerned with results on the computational complexity of PTP for several special cases as well as the approximation algorithms. The derivation of managerial implications based on a computational study is presented in Section 6. The paper closes with a summary and conclusions in Section 7.

2. Notation, Problem Description, and Definitions

PTP is defined as follows. We are given two kinds of vehicles: a set S of s homogeneous drones (small vehicles) and a single shuttle (large vehicle) with capacity k. The shuttle solely makes round trips between a depot and a launching point. Starting from the depot, the shuttle takes up to k drones on board and delivers them to the launching point. From there, the empty shuttle returns back to the depot. We assume that it takes $d \in \mathbb{N}$ time units to drive from the depot to the launching point or back. Hence, a complete round trip has a duration of $2d$. Note that – to derive a basic problem setting – we neglect all other times beyond travel, e.g., the time needed to load the shipments onto the drones, to load the drones onto the shuttle, or the time needed for swapping batteries of returning drones. We assume that these times are either negligible compared to the travel times or can be added to the latter.

Furthermore, we are given a set J of n jobs. Each job $j \in J$ is associated with the delivery of a shipment to a specific customer. Due to weight and size restrictions (Agatz et al. 2018), each drone can only carry a single shipment for a single customer at a time. Each drone delivery starts at the launching point and takes the flight time from there to the respective customer, the unloading time at the customer, and the time for the final flight leg from the customer back to the depot. In total, this takes a processing time $p_j \in \mathbb{N}$ for the drone assigned to the shipment to customer $j \in J$. Once returned to the depot, a drone is readily available for the next shipment. Note that, if $p_j < d$ for some $j \in J$, the drone assigned to job j will return to the depot before the shuttle returns. Thus, the drone inevitably has to wait for the shuttle. We can therefore set $p_j = d$ to compensate the waiting time. Therefore, without loss of generality, we assume $p_j \geq d$ for all $j \in J$.

In order to ease the notation in the remainder of this article, we define $h = \lfloor s/k \rfloor$, $k' = s - hk$, and $D = 2d$, where h represents the minimum number of round trips with k drones that can be performed consecutively at the beginning of a schedule (see Definition 2.1). The execution of these trips is part of what we will later define as the non-wasting property (see Definition 2.3). The notation is summarized in Table 1.

In our informal problem characterization, we have elaborated that PTP decides on the assignment of drones to multiple successive shuttle trips, each being defined by a departure time, and an assignment of customers to the drones of the trips. In fact, it is sufficient to only determine the departure times and the customers serviced in the corresponding trips.

Table 1: Notation used throughout the paper

S	set of drones (small vehicles)	$S = \{1, \ldots, s\}$
J	set of jobs (customers)	$J = \{1, \ldots, n\}$
p_j	processing time of job $j \in J$	$p_j \geq d \; \forall j \in J$
d	one-way travel time of shuttle (large vehicle)	
$D = 2d$	time needed for a round trip of the shuttle	
k	capacity of the shuttle (in number of drones)	
$h = \lfloor s/k \rfloor$	auxiliary notation	
$k' = s - hk$	auxiliary notation	

Definition 2.1 (Schedule)**.** *A schedule σ consists of a set of shuttle departure times* $\mathrm{T} = \{\tau_1, \tau_2, \ldots\}$ *and, for each $\tau \in \mathrm{T}$, a set of corresponding jobs $J(\tau) \subseteq J$ that are serviced on the respective round trip. For each $\tau \in \mathrm{T}$, $\eta(\tau) = \sum_{\tau' \in \mathrm{T}:\tau' < \tau} |\{j \in J(\tau') | p_j > \tau - d - \tau'\}|$ denotes the number of drones that are still on the way and therefore not available for being loaded onto the shuttle at time τ. Similarly, $\hat{\eta}(\tau) = \sum_{\tau' \in \mathrm{T}:\tau' < \tau} |\{j \in J(\tau') | p_j = \tau - d - \tau'\}|$ is the number of drones that return to the depot exactly at time τ.*

The prerequisites of a schedule being feasible are as follows.

Definition 2.2 (Feasible schedule)**.** *Schedule σ is feasible, if and only if*

1. *all customers are serviced, i.e., $J = \bigcup_{\tau \in \mathrm{T}} J(\tau)$,*
2. *the shuttle has enough time to return to the depot in between two round trips, i.e., $|\tau - \tau'| \geq D$ for all $\tau, \tau' \in \mathrm{T}$, $\tau \neq \tau'$, and*
3. *the number of drones loaded onto the shuttle does neither exceed the shuttle's capacity nor the number of drones currently available at the depot, i.e., $|J(\tau)| \leq \min\{k, s - \eta(\tau)\}$ for all $\tau \in \mathrm{T}$.*

Among all feasible schedules, PTP seeks a schedule σ with minimum makespan $C_{\max}(\sigma) = \max_{\tau_i \in \mathrm{T}, j \in J(\tau_i)}\{\tau_i + d + p_j\}$.

We now define a specific type of schedule that restricts the shuttle departure times and the number of drones loaded in each round trip.

Definition 2.3 (Non-wasting schedule)**.** *A feasible schedule σ is called non-wasting if and only if the following conditions hold for all $\tau \in \mathrm{T}$:*

1. *Either* $\max\{0, \tau - 2d\} \in \mathrm{T}$ *or* $\tau \in \bigcup_{\tau' \in \mathrm{T}}\{\tau' + d + p_j \mid j \in J(\tau')\}$.
2. *If* $\max\{0, \tau - 2d\} \notin \mathrm{T}$, *then* $\hat{\eta}(\tau) > s - |J(\tau)| - \eta(\tau)$.
3. $|J(\tau)| = \min\{k, s - \eta(\tau), n - \sum_{\tau' \in \mathrm{T}:\tau' < \tau} |J(\tau')|\}$.

The first property of Definition 2.3 ensures that the shuttle departs for a round trip immediately after having returned to the depot or that it departs exactly when some drones have returned. The latter case is further restricted by the second property, which enforces at least one of the drones that have just returned is loaded onto the shuttle for the next round trip. The third property ensures that the maximum possible number of drones (or the exact number of drones needed to serve the remaining customers) is loaded onto the shuttle for each round trip. It is easy to see that any optimal schedule can be transformed into a corresponding non-wasting schedule without increasing the makespan. Hence, in what follows, we will restrict ourselves to considering non-wasting schedules.

When designing algorithms for PTP, one can, for example, construct and combine strategies for generating shuttle departure times in non-wasting schedules on the one hand and procedures for assigning jobs to the drones of the respective round trips on the other hand. With respect to generating shuttle departure times, we will later consider two specific strategies, that are defined as follows.

Definition 2.4 (Conservative and aggressive strategies)**.** *In a conservative strategy, the (non-wasting) schedule is such that the shuttle departs as soon as at least k drones are available for being loaded or when the required number of drones that are needed to serve all customers that have not yet been served can be loaded. In an aggressive strategy, the shuttle departs whenever there is at least one available drone as well as a customer that remains to be served.*

The conservative strategy models the policy of transporting as many drones as possible in each round trip, while the aggressive strategy makes the shuttle depart for its trips as soon as possible. Both strategies may be advantageous under different circumstances. When s is rather small and d is large when compared with the job processing times, the conservative strategy may be a good choice. On the other hand, if d is relatively small and s is rather large, then one would expect the aggressive strategy to perform well.

3. Integer Program

Let t_{max} be some upper bound on the latest departure time of the shuttle in an optimal solution and let variables $s_t \in \mathbb{N}$, $t \in \{0, \ldots, t_{max}\}$, represent the number of drones that are not available immediately after the shuttle has left at time instant t as they have not yet returned from their last departure. Note that time instant $t = 0$ represents the beginning of the planning horizon. Additionally, define the following binary variables:

$$v_t := \begin{cases} 1 & \text{if a round trip starts at time instant } t \\ 0 & \text{else} \end{cases} \qquad \forall\, t \in \{0, \ldots, t_{max}\},$$

$$a_{j,t} := \begin{cases} 1 & \text{if customer } j \text{ is served by a drone loaded onto the shuttle} \\ & \text{starting a round trip at time instant } t \\ 0 & \text{else} \end{cases} \qquad \forall\, j \in J, t \in \{0, \ldots, t_{max}\}.$$

Then, an integer program for PTP is as follows.

$$\begin{aligned}
\min \quad & C_{max} && && (1)\\
\text{s.t.} \quad & t \cdot a_{j,t} + d + p_j \le C_{max} && \forall j \in J, t \in \{0, \ldots, t_{max}\}, && (2)\\
& \sum_{t' \in \{t, \ldots, t+D-1\}} v_{t'} \le 1 && \forall t \in \{0, \ldots, t_{max} - D + 1\}, && (3)\\
& \sum_{t \in \{0, \ldots, t_{max}\}} a_{j,t} = 1 && \forall j \in J, && (4)\\
& a_{j,t} = 0 && \forall j \in J, t \in \{-d - p_j, \ldots, -1\}, && (5)\\
& s_t = s_{t-1} + \sum_{j \in J} a_{j,t} - \sum_{j \in J} a_{j,t-d-p_j} && \forall t \in \{0, \ldots, t_{max}\}, && (6)\\
& s_{-1} = 0 && && (7)\\
& \sum_{j \in J} a_{j,t} \le k \cdot v_t && \forall t \in \{0, \ldots, t_{max}\}, && (8)\\
& s_t \le s && \forall t \in \{0, \ldots, t_{max}\}, && (9)\\
& v_t \in \{0, 1\} && \forall t \in \{0, \ldots, t_{max}\}, && (10)\\
& s_t \in \mathbb{N} && \forall t \in \{-1, \ldots, t_{max}\}, && (11)
\end{aligned}$$

$$a_{j,t} \in \{0,1\} \qquad \forall j \in J, t \in \{-d-p_j,\dots,t_{max}\}, \tag{12}$$

The objective function (1) minimizes the makespan, which is bounded from below by restrictions (2). Note that, as we assume that $p_j \geq d$ for all $j \in J$, we do not have to explicitly consider the last return of the shuttle to the depot in the latter restrictions. Constraints (3) ensure that at least D time units elapse between the start of two succeeding round trips. Restrictions (4) enforce every customer to be served by exactly one drone. Constraints (5) define auxiliary variables and fix them to zero. They are needed within constraints (6) and (7), that set the variables s_t, $t \in \{0,\dots,t_{max}\}$, and the auxiliary variable s_{-1} to their correct values. Note that the latter two terms on the right hand side of constraints (6) correspond to the number of drones that depart and land at the depot at time t, respectively, and this aligns with constraints (5). Restrictions (8) assure that at most k drones are loaded onto the shuttle for each round trip. Finally, constraints (9) model the fact that the number of drones is limited and restrictions (10)–(12) define the domains of the variables.

4. Computational Complexity

For the extreme case where $d = 0$, PTP is equivalent to $P||C_{max}$, i.e., the classical problem of scheduling jobs on s parallel machines to minimize the makespan, which is known to be strongly NP-hard and has been extensively studied for decades (Chen et al. 2014; Garey & Johnson 1978). The equivalence is due to the fact that, for $d = 0$, the shuttle capacity essentially does not restrict the number of drones that can be transported to the launching point by the shuttle as it can perform an arbitrary number of round trips within zero time units. Hence, each drone can immediately be transported to the launching point to start processing a new job after it has returned to the depot. As a result, PTP reduces to assigning the jobs to the drones so as to minimize the maximum load of the drones. When interpreting the drones as parallel machines, this corresponds to $P||C_{max}$. Hence, we have shown that PTP is strongly NP-hard. In this section, we will additionally show that PTP remains strongly NP-hard when restricted to the case $s = 2$ and $k = 1$ or the case $s = k = 2$. This is an interesting property in light of the fact that $P||C_{max}$ on a fixed number of two machines, i.e., the problem usually denoted by $P2||C_{max}$, is only weakly NP-hard as it can be solved in pseudo-polynomial time by dynamic programming (Karp 1972).

The general case with positive d is more representative for our problem setting, because it reflects the fact that some drones may have to wait for other drones so that they can be transported to the launching point by the shuttle in a single round trip. This type of synchronization property is also relevant for many other applications in machine scheduling. For example, in reliable machine scheduling, the machines need to be calibrated periodically to ensure high quality products. It is usually costly and time consuming to conduct these calibrations. However, there may be time savings when groups of machines are calibrated at the same time, so that some machines might be intentionally set to be idle or turned off in order to wait for other machines and, thus, synchronize the calibrating operations.

In light of the above synchronization property, we now define a variant of $P||C_{max}$.

Definition 4.1 (Synchronized Makespan Minimization Scheduling Problem)**.** *Given integers k and d, the Synchronized Makespan Minimization Scheduling Problem (SMMSP) is a special case of $P||C_{max}$ on some job set $\tilde{J}$ with job processing times $\tilde{p}_j$, $j \in \tilde{J}$, where a feasible solution must fulfill the following properties:*

1. *The number of jobs that share the same starting time is at most k.*

2. *The difference of any pair of two distinct job starting times is at least $2d = D$.*

Obviously, SMMSP with jobs $\tilde{J} = J$ and $\tilde{p}_j = p_j + d$ for $j \in \tilde{J}$ is equivalent to PTP. Adopting the classical three-field notation in Graham et al. (1979), we denote SMMSP by $P|sync \leq k, diff \geq D|C_{max}$. When the number s of parallel machines is fixed, we write $Ps|sync \leq k, diff \geq D|C_{max}$.

In order to prove the aforementioned complexity results, we will investigate two variants of SMMSP. In the first variant, we fix $k = 1$. This class of problems, $P|sync \leq 1, diff \geq D|C_{max}$, corresponds to a variant of $P||C_{max}$, where the difference of any pair of two job starting times must be at least D. In the second variant, $P|sync \leq s, diff \geq D|C_{max}$, we set $k = s$.

Our proofs are based on reductions of the 3-Partition Problem, which is well known to be strongly NP-hard (Garey & Johnson 1979).

Definition 4.2 (3-Partition Problem)**.** *Given positive integers b, c and a set of $3b$ positive integers $Q = \{a_1, a_2, \ldots, a_{3b}\}$ with $\sum_{i=1}^{3b} a_i = bc$, the 3-Partition Problem is to decide whether the set Q can be partitioned into b subsets $Q_1, Q_2, \ldots, Q_b$, such that each subset contains three elements and the sum of the numbers in each subset equals c.*

Without loss of generality, we will from now on restrict ourselves to 3-Partition instances with $a_i \in (0, c)$ for all $a_i \in Q$. Moreover, if some 3-Partition instance is a yes-instance, we will refer to a corresponding partitioning as a solution of this instance.

4.1. Strong NP-hardness for the Case $s = 2$ and $k = 1$

We will now show that $P|sync \leq 1, diff \geq D|C_{max}$ is strongly NP-hard even for fixed $s = 2$ and $D > 0$, i.e., we will prove strong NP-hardness of $P2|sync \leq 1, diff \geq D|C_{max}$. This result implies the fact that PTP is strongly NP-hard for the case $s = 2$ and $k = 1$.

Theorem 4.1. *$P2|sync \leq 1, diff \geq D|C_{max}$ is strongly NP-hard.*

Proof. Given an instance of 3-Partition, we construct an instance of $P2|sync \leq 1, diff \geq D|C_{max}$ as follows. Define $\ell = 10c$, and $D = 2c$. With each $a_i \in Q$, we associate a job with processing time $p_i = a_i + 3c$. We refer to these jobs as regular jobs and denote the set of these jobs by R. In addition, we create $b - 1$ "long" dummy jobs with identical processing time $\ell + 2D = 14c$ and one "short" dummy job with processing time $\ell + D = 12c$. Hence, there are $4b$ jobs in total. Let I denote the set of all of these jobs.

As $a_i \in (0, c)$, we have $p_i \in (3c, 4c)$ or, equivalently, $p_i \in (D + c, 2D)$ for each regular job $i \in R$. Furthermore, note that the sum of the processing times of any two regular jobs is smaller than $\ell - D$, the sum of the processing times of any three regular jobs is larger than $\ell - D$, and the sum of the processing times of any four regular jobs is larger than ℓ. Moreover, if the sum of processing times of any three regular jobs equals ℓ, then the sum of the corresponding three elements of Q equals c and vice versa.

We will now show that, if there is a solution to the 3-Partition instance, then there is a solution of the $P2|sync \leq 1, diff \geq D|C_{max}$ instance with makespan $b(\ell + D) = 12bc$. To see this, note that the sum of the processing times of the regular jobs that correspond to the elements of any subset of the 3-Partition instance equals $\ell = 10c$. Construct a schedule by allocating the long dummy jobs to the two machines in an alternating manner with starting time $(D + \ell)(i - 1) = 12c(i - 1)$ for the i-th job. Similarly, allocate the short dummy job to a machine so that it starts at time instant $(D + \ell)(b - 1) = 12c(b - 1)$. Note that, in this case, the completion time of the short dummy job equals $b(\ell + D) = 12bc$. As by definition of $P2|sync \leq 1, \mathit{diff} \geq D|C_{max}$, the difference of any pair of two job starting times must be at least D, every remaining available time interval on the two machines that results in a makespan of exactly $b(\ell+D) = 12bc$, has a length of exactly ℓ time units (see Figure 3), so that we can assign the regular jobs that correspond to the elements of the subsets of the partitioning

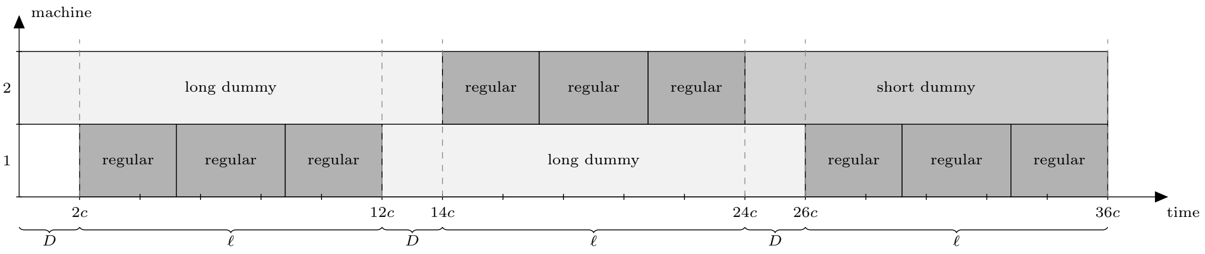


Figure 3: Illustration of the construction of an optimal schedule for $b = 3$

to these disjoint time intervals, which proves the above claim.

It remains to prove that, if the instance of $P2|sync \leq 1, \mathit{diff} \geq D|C_{max}$ has a solution with makespan $b(\ell+D)$, then the 3-Partition instance has a solution. Note that, if the makespan of a feasible solution of the instance of $P2|sync \leq 1, \mathit{diff} \geq D|C_{max}$ is $\frac{D+\sum_{j\in I} p_j}{2} = b(\ell+D)$, then there is no idle time on any of the machines except for the very beginning of the time horizon, because the difference of any pair of two job starting times must be at least D. As a consequence, the time instant at which the machines finish processing their last jobs is identical.

Denote the starting time and the completion time of job j in the considered schedule by t_j and C_j, respectively. Then, for all regular jobs $j \in R$, the time interval (t_j, C_j) contains no starting time of any other job, because $D < p_j < 2D$ and any starting time $t_{j'} \in (t_j, C_j)$ on the other machine would result in a violation of the property of a minimum difference of at least D time units of $t_{j'}$ to t_j, or to the starting time of the job which is scheduled immediately after j. If j is the last job on its machine, then $C_j = b(\ell + D)$ and $t_{j'} \in (t_j, C_j)$ would result in a makespan larger than $b(\ell + D)$.

Thus, the processing intervals of regular jobs are pairwise disjoint and one of the machines completes with the short dummy job as the last job. Define a block as a set of regular jobs scheduled successively on one machine without any idle time. Assume a block is maximal, i.e., the jobs immediately preceding and following the block are not regular, so that any two blocks are separated by at least D time units. Then, because the schedule contains no idle time besides D time units before the first job on only one of the machines, a dummy job is processed on the other machine while a block is processed on one machine. The dummy job's starting time is at least D time units earlier than the start of the block, and its completion time is at least D time units later than the completion time of the block, except for the dummy job scheduled last. Thus, the total processing time of a block is at most ℓ and it includes at most 3 jobs. There are b dummy jobs. Thus, there are at most b blocks. The total processing time of all regular jobs is $10bc = b\ell$. With less than b blocks, one of them would have a processing time larger than ℓ. As a consequence, the processing time of each block equals ℓ and we have a solution of the 3-Partition instance. This concludes the proof. $\square$

4.2. Strong NP-hardness for the Case $s = k = 2$

In this section, we will show that $P|sync \leq s, \mathit{diff} \geq D|C_{max}$ is strongly NP-hard even for fixed $s = 2$ and $D > 0$. It follows readily that PTP is strongly NP-hard for the case $s = k = 2$.

Within the proof, we will restrict our attention to instances of 3-Partition where $c/4 < a_i < c/2$ for all $a_i \in Q$ and any two elements of set Q have distinct values. The 3-Partition Problem remains strongly NP-hard under these assumptions (see Garey & Johnson (1979) for the first property and Hulett et al. (2008), Corollary 7, for the second property).

Theorem 4.2. *$P2|sync \leq 2, diff \geq D|C_{max}$ is strongly NP-hard.*

Proof. Given an instance of 3-Partition, we create an instance of $P2|sync \leq 2, diff \geq D|C_{max}$ as follows. Set $D = c/4$ and create a job i with processing time $p_i = a_i$ for each $a_i \in Q$. We refer to the corresponding jobs as regular jobs and denote the set of regular jobs by R. Additionally, create b dummy jobs with processing time c, so that there are $4b$ jobs in total.

We will first show that, if there is a solution $Q_1, Q_2, \ldots, Q_b$ to the 3-Partition instance, then there is a solution of the $P2|sync \leq 2, diff \geq D|C_{max}$ instance with makespan bc by scheduling the regular jobs on one of the machines in any order within their sets $Q_1, Q_2, \ldots, Q_b$ and processing all dummy jobs on the other machine. There are no idle times on the machines. It is obvious that the resulting schedule is such that both machines finish processing their last job at time bc. Furthermore, for each dummy job, there exists a regular jobs that starts at the same time instant. Hence, the difference of any pair of two distinct job starting times is at least $\min_{a_i \in Q} a_i > c/4 = D$ and we have a solution to the $P2|sync \leq 2, diff \geq D|C_{max}$ instance.

Now, assume there exists a solution of the $P2|sync \leq 2, diff \geq D|C_{max}$ instance with makespan bc. Note that, within this schedule, there is no idle time and both machines finish processing their last job at the same time instant bc. In what follows, we will prove that - in this case - the 3-Partition instance has a solution. We will use the notation introduced in the proof of Theorem 4.1.

First, observe that in the considered solution of the $P2|sync \leq 2, diff \geq D|C_{max}$ instance, there exists no pair of regular jobs with overlapping processing intervals. To see this, assume to the contrary that there exists a pair $j, j' \in R$, so that j and j' are scheduled on different machines and their processing overlaps. Without loss of generality, assume that $t_j \leq t_{j'}$. If $t_j = t_{j'}$, we have $0 < |C_j - C_{j'}| < c/4$ by our general restrictions on the considered 3-Partition instances. Hence, as the schedule has no idle time, the two other jobs starting immediately when j and j' are completed violate the required starting time difference of D. Moreover, because all a_i are different and larger than $c/4$, neither j nor j' can be the last jobs on their machines, so that we have a contradiction. If, on the other hand, $t_j < t_{j'}$, we have $t_j + D \leq t_{j'}$. Moreover, $C_j > t_{j'}$, because the processing of j and j' overlaps. Since additionally $D = c/4 < p_i < c/2$ for all $i \in R$, it follows that $0 < C_j - t_{j'} < D$ and $C_j < C_{j'}$. Therefore, C_j must be the starting time of another job so that the SMMSP constraints are violated, which is a contradiction. Therefore, the above observation is true.

Let j and j' be a regular and a dummy job, respectively. Then, by the same arguments we conclude that $t_{j'} \notin (t_j, C_j)$ and $C_{j'} \notin (t_j, C_j)$. As the schedule has no idle time, the intervals of processing dummy jobs do not overlap. Whenever a dummy job j' is scheduled on a machine, a block of regular jobs is scheduled at the same time on the other machine. The starting time of the first job in this block is identical to $t_{j'}$. The completion time of the last job in the block equals $C_{j'}$. Due to the fact that each block contains exactly 3 regular jobs, we have a solution to the instance of 3-Partition. □

4.3. Polynomial-Time Solvability for Identical Processing Times

In this section, we assume that all jobs have the same processing time, i.e., $p_j = p$, $j \in J$. Then, PTP reduces to the problem of having to determine optimal shuttle departure times in non-wasting schedules. When making use of the shuttle departure strategies introduced in Definition 2.4, a few problem settings become trivial.

Proposition 4.1. *If the processing times are all equal, $p_j = p$ for all $j \in J$, the following holds:*

1. *If $k' = 0$ or $Dh \geq d + p$, applying the conservative strategy or the aggressive strategy results in an optimal schedule.*
2. *If $D(h + 1) \leq d + p$, using the aggressive strategy results in an optimal schedule.*

Proof. Consider the case $k' = 0$. Then, for both strategies, the number of drones that are available in the depot will always be a multiple of k. Hence, when restricting the solutions to non-wasting schedules, both strategies result in identical solutions, in which the shuttle transports the maximum number of possible (shuttle capacity) or needed (remaining customers to be served) drones to the launching point in every round trip. Moreover, these solutions are optimal, since the shuttle departs immediately when a drone is available.

Now, consider the case $Dh \geq d+p$. If $d+p = 0$, the claim is obvious, so that we restrict our attention to the case $d + p > 0$. Hence, we may assume that $h > 0$ and, therefore, $s \geq k$. By definition of h, there are at least k drones available when departing for the first h round trips for both strategies. At time Dh, the shuttle returns to the depot from round trip h. At this point of time, the drones launched in the first round trip have already returned, as $Dh \geq d+p$. The same reasoning applies for the later round trips of the shuttle. Consequently, both strategies result in optimal schedules that are structured as in the above case $k' = 0$.

Now, assume that $D(h+1) \leq d+p$ and that the aggressive strategy is applied. Again, consider the non-trivial case $d > 0$. Then the shuttle starts the $(h + 1)$-th round trip at time Dh with exactly k' drones. The drones that are transported to the launching point in the first round will not return to the depot later than the shuttle returns from round trip h, because $Dh < d+p$. Hence, the resulting schedule is such that the maximum number of drones departs from the depot, which is either k, k' or limited by the remaining customers, in each round trip. □

For the remaining case, $k' > 0$ and $Dh < d + p < D(h + 1)$, we derive the following property.

Proposition 4.2. *Consider fixed values of s and k such that $k' > 0$. Moreover, assume that $p_j = p$ for all $j \in J$ and $Dh < d + p < D(h + 1)$. Let $f(n)$ be the optimal makespan of an instance of PTP with n jobs. Then*

$$f(n) = \begin{cases} d + p + D(\lceil \frac{n}{k} \rceil - 1), & \text{if } 0 < n \leq s \\ \min\{f(n - hk) + (d + p), f(n - s) + D(h + 1)\}, & \text{if } n > s. \end{cases} \tag{13}$$

Proof. In case of $n \leq s$, i.e., when there are more drones than jobs, the shuttle will never have to wait for drones to arrive at the depot before departing for a round trip. Hence, applying the conservative strategy or the aggressive strategy results in an optimal schedule with $\lceil \frac{n}{k} \rceil$ round trips and makespan $d + p + D(\lceil \frac{n}{k} \rceil - 1)$.

Now, consider the case $n > s$. Note that $h > 0$ as $d + p < D(h + 1)$ and $p \geq d$. As we restrict our attention to non-wasting schedules, the optimal schedule is such that there are k drones transported by the shuttle in the first h round trips. Hence, $\tau_i = D(i - 1)$ for the i-th round trip, $i = 1, \ldots, h$. Moreover, for $h \geq 2$, the time interval between the return of the drones launched in round trips $i \in \{1, \ldots, h - 1\}$ and $i' = i + 1$ has a length of exactly D time units. In what follows, we will differentiate between two cases.

First, assume that the optimal schedule is such that $\tau_{h+1} > \tau_h + D$, i.e., the shuttle waits for drones to return to the depot before departing for round trip $h+1$. As the k drones launched in the first round trip will return to the depot at time $d + p$, we have $\tau_{h+1} = d + p$. Hence, we have reduced the considered problem to a subproblem with $n - hk$ jobs and starting time $d + p$, so that $f(n) = f(n - hk) + (d + p)$.

Now, assume that $\tau_{h+1} = \tau_h + D$, so that there are k' drones launched in round trip $h+1$ in an optimal schedule. As $d + p < D(h+1)$, the drones in the first round will have returned to the depot before the shuttle returns from round trip $h+1$. Hence, $\tau_{h+2} = \tau_{h+1} + D = D(h+1)$.

As a result, we have reduced the considered problem to a subproblem with starting time $D(h+1)$ and $n - s$ jobs, so that $f(n) = f(n-s) + D(h+1)$. $\square$

Proposition 4.2 characterizes a dynamic programming (DP) approach for solving PTP when all processing times are identical and when $k' > 0$ and $Dh < d + p < D(h+1)$. The runtime of this DP approach is linear in the number of jobs. If n is a multiple of the least common multiple of s and $s - k'$ then the optimal solution can be determined directly, either conservatively or aggressively. If n is not larger than the least common multiple of s and $s - k'$, the runtime can be improved to $O(s)$. To see this, we restrict our attention to the case $n > s$ and assume that the optimal schedule is such that the first option in the minimization of Equation (13) is chosen x times and the second option is chosen y times. Then the makespan of the optimal schedule equals

$$x(d+p) + yD(h+1) + (d+p) + D(\left\lceil \frac{z}{k} \right\rceil - 1),$$

where $1 \le z \le s$ is such that $n = x \cdot (s - k') + y \cdot s + z$. When x is fixed, we have $y \le \left\lfloor \frac{n-1-x(s-k')}{s} \right\rfloor$ and $z = n - x(s-k') - ys$. As we assume that n is not larger than the least common multiple of s and $s - k'$, we have $x \le \left\lfloor \frac{n}{s-k'} \right\rfloor \le s$, so that the runtime of the DP reduces to $O(s)$ as claimed above.

As a consequence of the results presented in this section, we can conclude that PTP is polynomial time solvable when all processing times are identical.

5. Approximation Algorithms

In this section, we propose approximation algorithms for different cases of PTP. An algorithm A is said to be a *ρ-approximation*, $\rho \ge 1$, if $A(I) \le \rho \cdot OPT(I)$ for all problem instances I, where $A(I)$ and $OPT(I)$ are the objective values obtained by solving I using A and the optimal objective value, respectively. An *approximation scheme* is a family of algorithms that allows the user to specify any level $\epsilon > 0$ of accuracy of the approximation and includes an algorithm A_ϵ for each ϵ, such that A_ϵ is an $(1+\epsilon)$-approximation algorithm. As might be expected, the runtime increases as more accurate levels are requested. A *polynomial-time approximation scheme* (PTAS) is an approximation scheme with polynomial runtime in the input size for fixed ϵ.

In Section 5.1, we assume that the shuttle capacity is unbounded. For this case, we show that a well known list scheduling approach is a 2-approximation. In Section 5.2, we then turn our attention to general values of the shuttle capacity, but assume that the set of jobs is restricted to jobs with "small" processing times. We present a PTAS for this case. These results are utilized in Section 5.3, where we conclude our deliberations on approximation algorithms with a $(1+\epsilon)(3+\epsilon)$ approximation algorithm for the general case of PTP.

5.1. Unbounded Shuttle Capacity

Let us assume that the capacity of the shuttle is sufficiently large for loading all drones, i.e., $k = s$. For this case, we propose a greedy algorithm as a 2-approximation. It is based on adjusting the input by rounding the processing times of the jobs. More specifically, we create an instance of $Ps||C_{max}$ as follows. For each $j \in J$, we create a job (referred to as a *rounded job*) with processing time $p_j^{\#} = 2^\theta D$, where $\theta \in \mathbb{N}$ is the smallest positive integer such that $2^{\theta-1}D < p_j + d \le 2^\theta D$. Note that $p_j^{\#} < 2(p_j + d)$.

For the rounded jobs, the processing time $p_j^{\#}$ is a multiple of D. As a result, for any feasible solution of the instance of $Ps||C_{max}$ with rounded jobs that has no idle time between two consecutive jobs on the same machine, the starting time and completion time of each job must be a multiple of D. Therefore, and because $k = s$, a feasible solution of the instance of $Ps||C_{max}$ is also feasible to our PTP instance. Now, consider an optimal solution of $Ps||C_{max}$ and assume its makespan C_{max}^P is attained at machine i (corresponding to drone i). Let J_i be the set of all jobs scheduled on this machine. Then, we obtain $C_{max}^P = \sum_{i \in J_i} p_i^{\#} < 2 \sum_{i \in J_i}(p_i + d) \leq 2C_{max}^{PTP}$, where C_{max}^{PTP} is the optimal objective function value to our instance of PTP. The strict inequality holds because $p_j^{\#} < 2(p_j + d)$ for each job $j \in J$. Thus, an optimal solution to the instance of $Ps||C_{max}$ is a 2-approximation to our PTP instance.

The remaining part of this section is to find an optimal solution for the instance of $Ps||C_{max}$ with rounded jobs. We assume that the jobs are sorted and indexed in non-increasing order of their processing times (Longest Processing Time, LPT, order).

The algorithm *Best-Fit* for $Ps||C_{max}$ is a classical (greedy) list scheduling approach that iteratively assigns jobs to machines. It considers the jobs in their LPT order and assigns them to a machine with the smallest current workload without idle time (see Algorithm 1). Here, the workload of a machine is defined by the sum of

Algorithm 1. Best-Fit

```
  Input: Instance of $Ps||C_{max}$
  Output: Schedule $\Gamma_{\bar{s}}$ for all machines $\bar{s} \in S$
1 Initialize $\tau_{\bar{s}} := 0$ for all machines $\bar{s} \in S$;            ▷ Initialize workload of all machines
2 Initialize $\Gamma_{\bar{s}} := [\ ]$ for all machines $\bar{s} \in S$;     ▷ Initialize empty schedules for all machines
3 for $j = 1$ to $n$ do                                                   ▷ Iterate over jobs in LPT order
4     Select $\tilde{s} \in \arg\min_{\bar{s} \in S} \tau_{\bar{s}}$;                 ▷ Select machine with smallest workload
5     $\tau_{\tilde{s}} := \tau_{\tilde{s}} + p_j$;                                              ▷ Update workload
6     $\Gamma_{\tilde{s}} := [\Gamma_{\tilde{s}}, j]$;                                 ▷ Append job to the machine's schedule
7 end
```

the processing times of all jobs that have already been assigned to the machine.

Lemma 5.1. *Best-Fit is optimal for $Ps||C_{max}$ with rounded jobs and, thus, a 2-approximation to PTP for the case $k = s$.*

Proof. Consider a Best-Fit schedule σ for our instance of $Ps||C_{max}$ with rounded jobs on s machines. We assume that $p_1^{\#} \geq p_2^{\#} \geq \cdots \geq p_n^{\#}$. Let j' be the smallest index of a job that is completed at the final makespan of σ. Furthermore, let $J' = \{1, 2, \ldots, j'\}$ be the subset of the largest j' rounded jobs and t' be the starting time of job j' in σ. Let $C_{max}(J')$ be the makespan of an optimal schedule for the jobs in J'. Then, we have $C_{max}(J') > t'$, because $s \cdot C_{max}(J') > \sum_{i \in J' \setminus \{j'\}} p_i^{\#} \geq s \cdot t'$. The first inequality results from the fact that each machine completes processing of its jobs within $C_{max}(J')$ and the total processing time on all s machines is not more than $\sum_{i \in J'} p_i^{\#}$. The second inequality estimates the total workload of s machines. None of them completes the subset of jobs from J' assigned to it before t', otherwise Best-Fit would have assigned job j' to a machine completing before t'. For rounded jobs, the processing time of each job of J' is a multiple of $p_{j'}^{\#}$. Therefore, both $C_{max}(J')$ and t' are multiples of $p_{j'}^{\#}$, which implies that $C_{max}(J') \geq p_{j'}^{\#} + t'$, because of $C_{max}(J') > t'$. Due to the fact that the schedule obtained by the Best-Fit algorithm has makespan $p_{j'}^{\#} + t'$, we conclude that Best-Fit gives an optimal schedule. ☐

5.2. PTAS for Small Jobs

Recall that the problem $Ps|sync \leq k, diff \geq D|C_{max}$ with jobs $\tilde{J} = J$ and $\tilde{p}_j = p_j + d$ for $j \in \tilde{J}$ is equivalent to PTP. In this section, we consider instances of $Ps|sync \leq k, diff \geq D|C_{max}$ on s machines, where for a constant

$\epsilon > 0$, we assume that the input consists only of *small* jobs j with $D \leq \tilde{p}_j < D/\epsilon$ for $j \in \tilde{J}$. Note that the first inequality is equivalent to our general assumption $p_j \geq d$. We will provide a corresponding PTAS based on classical geometric rounding.

For each job $j \in \tilde{J}$, we round the processing time to the value $p_j^{\#} = D(1+\epsilon)^i$ with $D(1+\epsilon)^{i-1} < \tilde{p}_j \leq D(1+\epsilon)^i, i \in \mathbb{N}$. Note that, in contrast to the previous subsection, we round the values $\tilde{p}_j$ rather than the values p_j. Let $J^{\#}$ denote the set of jobs with rounded processing times. Again, we will refer to these jobs as *rounded jobs*. The number of distinct values of processing times for rounded jobs is bounded by the i that satisfies $D(1+\epsilon)^i \geq D/\epsilon$. We will refer to this specific i as v, so that $v = \lceil \log_{1+\epsilon} 1/\epsilon \rceil$. Due to the fact that $p_j^{\#} \leq (1+\epsilon)\tilde{p}_j$ for $j \in \tilde{J}$, optimally solving the instance with rounded jobs yields an $(1+\epsilon)$-approximation solution.

The rounded jobs instance can be optimally solved by dynamic programming as follows. Let $\Psi = \{\sum_{i=0}^{v} x_i \cdot D(1+\epsilon)^i \mid x_i \in \mathbb{N}_{\geq 0}, \sum_{i=0}^{v} x_i < n\}$. According to the non-wasting property of Definition 2.3, the set Ψ contains every possible shuttle departure time. $|\Psi|$ is bounded by n^{v+1} because of the above rounding procedure. Moreover, from $D(1+\epsilon)^{i-1} < \tilde{p}_j < D/\epsilon$, we conclude that $p_j^{\#} = D(1+\epsilon)^i < D(1+\epsilon)/\epsilon$. Thus, for each job $j \in J^{\#}$, there are at most $\lceil (1+\epsilon)/\epsilon \rceil = \lceil 1/\epsilon \rceil + 1$ shuttle departures (including the one that carries job j) during the execution of the job j with processing time $p_j^{\#}$ in the optimal schedule. Then, in the dynamic program, it is sufficient to record the information of the most recent $\lceil 1/\epsilon \rceil + 1$ shuttle departures. Therefore, a state $\mathfrak{s}$ of the dynamic programming procedure is defined as $\mathfrak{s} = (J_0^{\#}, \tau_1, \tau_2, \ldots, \tau_{\lceil 1/\epsilon \rceil + 1}, J^{\#}(\tau_1), J^{\#}(\tau_2), \ldots, J^{\#}(\tau_{\lceil 1/\epsilon \rceil + 1}))$. It relates to a partial schedule in which the jobs $J_0^{\#}$ are completed with s drones and one shuttle of capacity k, given that in the most recent $\lceil 1/\epsilon \rceil + 1$ rounds of the partial schedule, the shuttle has fixed departure times $\tau_1 \geq \tau_2 \geq \cdots \geq \tau_{\lceil 1/\epsilon \rceil + 1}$ and, correspondingly, fixed job assignments $J^{\#}(\tau_1), J^{\#}(\tau_2), \ldots, J^{\#}(\tau_{\lceil 1/\epsilon \rceil + 1})$, where $J_0^{\#} \subseteq J^{\#}$, and for each $i \in \{1, 2, \ldots, \lceil 1/\epsilon \rceil + 1\}$, $\tau_i \in \Psi$, $J^{\#}(\tau_i) \subseteq J^{\#} \setminus J_0^{\#}$. For some state $\mathfrak{s}$, we denote the corresponding makespan by $f(\mathfrak{s})$ and the minimum makespan by $F(\mathfrak{s})$. The problem is divided into subproblems by testing the next shuttle departure times after time τ_1 and the corresponding job assignments. Then, if $J_0^{\#} = \emptyset$, we have

$$F(\emptyset, \tau_1, \tau_2, \ldots, \tau_{\lceil 1/\epsilon \rceil + 1}) = \max_{\tau_i \in \{\tau_1, \tau_2, \ldots, \tau_{\lceil 1/\epsilon \rceil + 1}\}, j \in J^{\#}(\tau_i)} \{\tau_i + p_j^{\#}\}.$$

Otherwise,

$$\begin{aligned} F(J_0^{\#}, \tau_1, \tau_2, \ldots, \tau_{\lceil 1/\epsilon \rceil + 1}) &= F(J_0^{\#} \setminus J', \tau_0, \tau_1, \tau_2, \ldots, \tau_{\lceil 1/\epsilon \rceil}) = \\ \min_{\tau_0 \in NEXT, J' \subseteq J_0^{\#}} & f(J_0^{\#} \setminus J', \tau_0, \tau_1, \tau_2, \ldots, \tau_{\lceil 1/\epsilon \rceil}, J', J^{\#}(\tau_1), J^{\#}(\tau_2), \ldots, J^{\#}(\tau_{\lceil 1/\epsilon \rceil})), \end{aligned}$$

where $NEXT = \{t \mid t \geq \tau_1 + D, t = \tau_i + D(1+\epsilon)^{\lambda}, \lambda \in \{0, 1, \ldots, v\}, i \in \{1, 2, \ldots, \lceil 1/\epsilon \rceil\}\}$, $|J'| \leq \min\{k, s - \eta(\tau_0)\}$, and $\eta(\tau_0) = \sum_{\tau' \in \{\tau_1, \tau_2, \ldots, \tau_{\lceil 1/\epsilon \rceil + 1}\}} |\{j \in J^{\#}(\tau') | p_j^{\#} > \tau_0 - \tau'\}|$. The partial schedule is feasible as the third property of Definition 2.2 holds.

The important fact of this dynamic programming approach is that any job of $J^{\#}(\tau_{\lceil 1/\epsilon \rceil + 1})$ will start at time $\tau_{\lceil 1/\epsilon \rceil + 1}$ and finish no later than $\tau_1 + D$ since $p_j^{\#} < D \cdot (\lceil 1/\epsilon \rceil + 1)$ for any $j \in J^{\#}$. Then, the approach is correct because we have enumerated over all possible next shuttle departure times τ_0 and the corresponding sets of jobs J'.

With respect to time complexity, note that any subset of $J^{\#}$ can be encoded as a vector with space complexity $O(n^{v+1})$ as there are at most $v+1$ distinct job processing times in $J^{\#}$. Similarly, each subset of jobs $J^{\#}(\tau_i)$ can be encoded as a vector with space complexity $O(k^{v+1})$ because $|J^{\#}(\tau_i)| \leq k$. The variables $\tau_1, \tau_2, \ldots, \tau_{\lceil 1/\epsilon \rceil}$

can be encoded together with space complexity $O(|\Psi|^{\lceil 1/\epsilon \rceil})$. Therefore, the table size of the dynamic program is $O(n^{v+1}k^{(v+1)\lceil 1/\epsilon \rceil} \cdot |\Psi|^{\lceil 1/\epsilon \rceil}) = O(n^{(v+1)(\lceil 1/\epsilon \rceil+1)}k^{(v+1)\lceil 1/\epsilon \rceil})$. It takes $O(\lceil 1/\epsilon \rceil(v+1) \cdot k^{v+1})$ time to enumerate all possible time τ_0 and corresponding sets of jobs $J^{\#}(\tau_0)$. Note that the term $\lceil 1/\epsilon \rceil(v+1)$ is a constant which only relies on ϵ, and providing that $k \leq n$, the final complexity of the dynamic programming approach is $O((nk)^{(v+1)(\lceil 1/\epsilon \rceil+1)})$, which is polynomial in n.

5.3. General Case

In this section, we consider the general case of PTP.

As in Section 5.1 we have presented a 2-approximation algorithm for the case $k = s$, we will first restrict ourselves to the case $k < s$. We slightly modify our previous definition of small jobs and partition the set of jobs into a subset of small jobs and another one with big jobs. Given a constant $\epsilon > 0$, job $j \in \tilde{J}$ is called *big* if $\tilde{p}_j > \xi$, otherwise *small*, where $\xi = \frac{D}{\min\{\epsilon, 1/h\}}$.

Recall our definition of $h = \lfloor s/k \rfloor$. Let $J_b \subseteq J$ be the subset that contains all big jobs, and let $J_s = J \setminus J_b$. We propose an algorithm which solves the subproblem consisting of only big jobs and the subproblem that contains only small jobs independently and then merge the two schedules to a final schedule.

Let us first consider the subproblem that solely consists of small jobs J_s. We can use the PTAS described in Section 5.2 if $\epsilon < 1/h$. If $\epsilon \geq 1/h$, we can find an optimal solution in linear time.

Lemma 5.2. *Assume that $k < s$ and that there are only small jobs. If $\xi = Dh$ then PTP can be optimally solved in linear time, otherwise there exists a PTAS.*

Proof. If $\epsilon \geq 1/h$ and, therefore, $\xi = Dh$, we propose a greedy algorithm that generates an optimal non-wasting schedule. It applies the conservative strategy and schedules the jobs in their LPT order. In order to prove optimality, we show that there will always be at least k drones available upon each return of the shuttle to the depot. For the first h rounds, the claim is true because $s \geq hk$. In round h, the shuttle returns at time $t = Dh$, and at this moment all k drones launched in the first round must already have returned because $\tilde{p}_j \leq \xi = Dh$ for any small job $j \in J_s$. Hence, the claim is true for round $h+1$. It is straightforward to use a similar argument to show that the claim is true for all rounds.

If otherwise $\epsilon < 1/h$ and, therefore, $\xi = D/\epsilon$, we apply the algorithm introduced in Section 5.2 to receive a PTAS for the problem with only small jobs. □

Now, assume that there are only big jobs J_b.

Lemma 5.3. *Assume that $k < s$ and that there are only big jobs. If there is an α-approximate solution for $P||C_{max}$, then there is a $(2+\epsilon)\alpha$-approximate solution for PTP.*

Proof. Consider an α-approximate solution for any instance of problem $P||C_{max}$ with processing time $\tilde{p}_j$ for job $j \in J_b$ on s machines. Based on this solution, we obtain a solution to our problem as follows. We keep the assignment of the jobs to the machines (drones), the order of the jobs on the same machine (drone), and apply the aggressive strategy with the policy that the drones are loaded on a first-come-first-serve basis. Consider an arbitrary drone i and a job $j \in J_b$ which is processed by the drone, let t be the completion time of job j (i.e., the time when the drone arrives at the depot after having executed the job), and let t' be the earliest possible shuttle departure time for a round trip that contains drone i such that $t' \geq t$. Then, we claim that $t' - t \leq Dh + D$. Suppose the opposite that $t' - t > Dh + D$. Let $\hat{t}$ be the earliest shuttle departure time in $[t, t')$.

Then, $\hat{t} < t + D$ and, during time interval $[\hat{t}, t')$, the shuttle departs at times $\hat{t}, \hat{t} + D, \hat{t} + 2D, \ldots, \hat{t} + w \cdot D$ (i.e., the shuttle departs immediately after its return). In each of these $w + 1$ round trips, the shuttle is full because drone i is available at time t and loaded onto the shuttle at time t', where $t \leq \hat{t} \leq \hat{t} + w \cdot D = t' - D$. We get $(w+1)k < s < (h+1)k$, i.e., $w \leq h - 1$, which contradicts the assumption that the shuttle departs at least $h+1$ times before drone i is loaded. Hence, $t' - t \leq t' - \hat{t} + D \leq D(h+1)$. That is $(t'-t)/\tilde{p}_j \leq (h+1) \cdot \min\{\epsilon, 1/h\}$ due to the fact that $j \in J_b$ and $\tilde{p}_j > \xi$. Moreover, $(h+1) \cdot \min\{\epsilon, 1/h\} \leq 1 + \epsilon$. Therefore, the lemma is proved. □

The schedules computed by the procedures introduced in Lemmas 5.2 and 5.3 can be merged as follows.

Lemma 5.4. *Assume that $k < s$. Given an α-approximation solution for small jobs J_s, and a β-approximation solution for big jobs J_b, merging these two solutions yields an $(\alpha + \beta)$-approximation solution for jobs J.*

Proof. Note that the optimal solution of the problem that contains small (resp. big) jobs J_b only, is a lower bound for the optimal solution of jobs J. Hence, the lemma is proved. □

We can now conclude this section with our main result with respect to approximation algorithms for the general case of PTP.

Proposition 5.1. *There exists an approximation algorithm with approximation ratio $(1+\epsilon)(3+\epsilon)$ for PTP.*

Proof. First, note that there exists a PTAS for $P||C_{max}$ for a constant number of machines (Graham 1969) and when the number of machines is part of the input (Hochbaum & Shmoys 1987). Then, based on the 2-approximation algorithm presented in Section 5.1 (case $k = s$) and Lemma 5.4 (case $k < s$), we receive an algorithm with approximation ratio $(1+\epsilon)(2+\epsilon) + (1+\epsilon) = (1+\epsilon)(3+\epsilon)$ for the general case of PTP. □

6. Computational Study and Managerial Implications

In this section, we provide decision support for transportation managers who have to set up a piggyback transportation process. In most applications, the main cost drivers will certainly be the capacity of the large vehicle and the fleet size of small vehicles. We therefore explore how to best allocate investments among these two cost drivers. To do so, we provide a heuristic approach for PTP. Specifically, we make use of our structural insights and provide a straightforward heuristic, which is kept easy and quick to implement, but leads to near-optimal solutions and is very fast. Throughout this section, we assume that the jobs are sorted and indexed in their LPT order.

Our heuristic approach is introduced in Section 6.1. Its computational performance is then evaluated in Section 6.2. The analysis of the managerial aspects indicated above is subject of Section 6.3. All computational tests were performed on a PC with an Intel® Core™ i7-4770 CPU, running at 3.4 GHz, with 16 GB of RAM under a 64-bit version of Windows 8. All algorithms were implemented in C^{++} (Microsoft Visual Studio Professional 2013). We used IBM ILOG CPLEX in version 12.7 as a MIP solver.

6.1. Basic Heuristic Approach

One of the main characteristics of PTP is the fact that it aims at solutions that synchronize the return of drones to the depot so that sufficiently large groups of drones can be transported to the launching point without making the shuttle wait at the depot for too long (synchronization property, see Section 4). This can be taken account of in simple heuristic approaches by first allocating jobs to distinct drones so that the resulting workload is well balanced among the drones, and afterwards generating round trips of the shuttle by deciding on shuttle departure times and the jobs that are associated to the trips. In the following, we make use of this idea.

In order to allocate the jobs to the drones so that the resulting workload is well balanced, we make use of the procedure Best-Fit (Algorithm 1 in Section 5.1). Based on the resulting drone schedules $\Gamma_{\bar{s}}$ for all $\bar{s} \in S$, we construct a non-wasting schedule as illustrated in Algorithm 2. The algorithm iteratively assigns the drones

Algorithm 2. Determine non-wasting schedule

```
Input: Instance of PTP, schedules $\Gamma_{\bar{s}}$ for all $\bar{s} \in S$, parameter $t_w$
Output: Departure times $\Phi$ with job sets $J(\tau) \subseteq J$ for all $\tau \in \Phi$, makespan $C_{max}$
1  Initialize $\Phi := \emptyset$;                                    ▷ Initialize empty set of round trip departure times
2  Initialize $available[\bar{s}] := 0$ for all $\bar{s} \in S$;        ▷ Initialize drone availability times
3  Initialize $shuttleTime := 0$;                                      ▷ Initialize shuttle availability time
4  Initialize $waitingTime := 0$, $\bar{J} := \emptyset$;              ▷ Initialize auxiliary variables for round trips
5  Initialize $jobCounter := 0$, $tripCounter := 0$;                   ▷ Initialize counters
6  do
7      Re-order drone set $S$;                                         ▷ Prioritize drones
8      $tripCounter := tripCounter + 1$;                               ▷ Consider next round trip
9      Initialize $\tau_{tripCounter} := 0$;                           ▷ Initialize round trip departure time
10     $\bar{J} := \emptyset$, $waitingTime := 0$;                     ▷ Update round trip variables
11     forall $\bar{s} \in S$ do                                       ▷ Iterate over prioritized drones
12         if $available[\bar{s}] \leq shuttleTime + t_w$ and $|\Gamma_{\bar{s}}| > 0$ then   ▷ Drone on time / jobs remaining?
13             Delete first element from $\Gamma_{\bar{s}}$ and insert it into $\bar{J}$;   ▷ Assign job to round trip
14             $jobCounter := jobCounter + 1$;                         ▷ Increment counter
15             if $available[\bar{s}] > shuttleTime$ and $available[\bar{s}] - shuttleTime > waitingTime$ then   ▷ Wait?
16                 $waitingTime := available[\bar{s}] - shuttleTime$;  ▷ Update waiting time for round trip
17             end
18         end
19         if $|\bar{J}| = k$ then break;                              ▷ Remaining capacity?
20     end
21     if $\bar{J} = \emptyset$ then                                   ▷ Empty round trip?
22         forall drones $\bar{s}$ with minimum value $available[\bar{s}]$ do   ▷ Wait for drone(s)
23             Delete first element from $\Gamma_{\bar{s}}$ and insert it into $\bar{J}$;   ▷ Assign job to round trip
24             $jobCounter := jobCounter + 1$;                         ▷ Increment counter
25             $waitingTime := available[\bar{s}] - shuttleTime$;      ▷ Update waiting time for round trip
26             if $|\bar{J}| = k$ then break;                          ▷ Remaining capacity?
27         end
28     end
29     $\tau_{tripCounter} := shuttleTime + waitingTime$, $\Phi := \Phi \cup \{\tau_{tripCounter}\}$;   ▷ Update departure time
30     Initialize $J(\tau_{tripCounter}) := \bar{J}$;                  ▷ Initialize job set of round trip
31     $shuttleTime := shuttleTime + D + waitingTime$;                 ▷ Update shuttle availability time
32     Update $available[\bar{s}]$ for all $\bar{s} \in S$;            ▷ Update drone availability times
33 while $jobCounter < n$;
34 Initialize $C_{max}$ with maximum value $available[\bar{s}]$, $\bar{s} \in S$;   ▷ Compute makespan
35 Return $C_{max}$, $\Phi$, and $J(\tau)$ for all $\tau \in \Phi$;   ▷ Terminate procedure
```

(or their respective jobs) to the round trips of the shuttle. During runtime, it keeps track of the time instants at which the drones return to the depot after having executed a job that has been assigned to a round trip (availability time, $available[\bar{s}]$ for drones $\bar{s} \in S$). Similarly, it dynamically updates the time at which the shuttle is available for executing its next round trip ($shuttleTime$). The algorithm uses a parameter t_w (shuttle waiting time), that defines the aspired maximum number of time units that the shuttle waits ($waitingTime$) at the depot for loading the drones of its next round trip (loop 11–20). Only if no drone has arrived within the last t_w time units, the waiting time can exceed the value of t_w, because empty round trips cannot improve a solution (lines 21–28). Hence, for $t_w = 0$, the algorithm applies the aggressive strategy. Note, however, that a very large value of t_w will not necessarily result in the conservative strategy, as the algorithm considers the drones in some specific order (indicated by the ordering routine of line 7 and in loop 11–20) that may differ from the order in which the drones arrive at the depot. The algorithm prioritizes drones with a larger remaining workload. Among two drones that have an identical remaining workload, the algorithm selects the drone with a smaller number of remaining jobs or, if this value is also identical, the drone with smaller availability time. As a last tie-breaker, the algorithm prioritizes the drone with the smallest index.

6.2. Performance of Heuristic Approach

In order to analyze the performance of our basic heuristic approach, we randomly generated a series of small test instances. Table 2 depicts the parameter values that these instances are based upon. The number of drones

Table 2: Parameter values of small test instances

s	n	k	d	λ
5, 7, 9, 11	$2s$, $4s$, $6s$, $8s$, $10s$	$\lfloor \frac{s}{2} \rfloor$, $\lfloor \frac{s}{3} \rfloor$, $\lfloor \frac{s}{4} \rfloor$	$[1, 10]$, $[1, 50]$, $[1, 100]$	0.2, 0.8, 1.5, 2

s varies from 5 to 11. The number of jobs n is a multiple of the number of drones. The corresponding factor ranges from 2 to 10. Similarly, the capacity k of the shuttle is set to some fraction of the number of drones. The values of the one-way travel time d were drawn from uniform distributions over the intervals given in the table. The processing times p_j of the jobs $j \in J$ were generated by drawing values x_j from an exponential distribution (density function $f(x) = \lambda \exp^{-\lambda x}$, $x \geq 0$) with parameter λ and then setting $p_j = d + d \cdot x_j$. This generation scheme is based on the assumption that the airship hovers directly over the middle of the city center, so that the majority of customers living in the innermost city have a processing time close to value d. Larger processing times are less likely, because the population density thins out toward the outskirts. This effect is steered by parameter λ, where a smaller (larger) value of λ makes larger processing times more (less) likely. For increasing vales of λ, the expected average processing times of the test instances decrease. We generated one instance for each combination of the parameter values. With respect to the capacity of the shuttle, however, we discarded all instances with $k < 2$. This resulted in a total of 540 test instances.

For each test instance, we called Algorithm 2 with four distinct parameter values $t_w \in \{0.2d, 0.5d, d, 1.5d\}$ and selected the best solution generated by these runs of the algorithm. Additionally, in order to provide a benchmark solution, we called CPLEX in its standard settings with a time limit of 900 seconds on the corresponding instance of model (1)–(12). Here, we used

$$t_{max} = \left\lfloor \frac{n}{\min\{s,k\}} \right\rfloor \cdot d + \sum_{i=1}^{\left\lfloor \frac{n}{\min\{s,k\}} \right\rfloor} \max_{j \in \{(i-1)\cdot\min\{s,k\}+1,\ldots,i\cdot\min\{s,k\}\}} p_j \tag{14}$$

as an upper bound on the latest departure time of the shuttle. When considering an example instance with $n = 3$, $s = k = 2$, $d = 1$ and $p_1 = p_2 = p_3 = 1$, it can be seen that (14) provides a tight bound on t_{max} for that particular case.

Consider an instance I of PTP, let $C_{\max}(sol)$ denote the makespan of some feasible solution sol of I, and denote the best corresponding solution determined by CPLEX by sol_C and the best solution determined by the four runs of Algorithm 2 based on the drone schedules computed with Algorithm 1 by sol_H. Then, we define the gap of this latter solution with respect to sol_C as

$$\frac{C_{\max}(sol_H) - C_{\max}(sol_C)}{C_{\max}(sol_C)}. \tag{15}$$

Note that we set $C_{\max}(sol_C) := \infty$ if CPLEX does not find a feasible solution.

The computational results are illustrated in Table 3. It is based on defining instance sets for all combinations of the values of parameters s, k, and n, and presents information on the percentage of instances of the corresponding set that CPLEX was able to find a feasible solution for (column "feas."), the percentage of instances that CPLEX solved to optimality (column "opt."), and the corresponding average runtime in seconds (column "t_{avg}"). For the

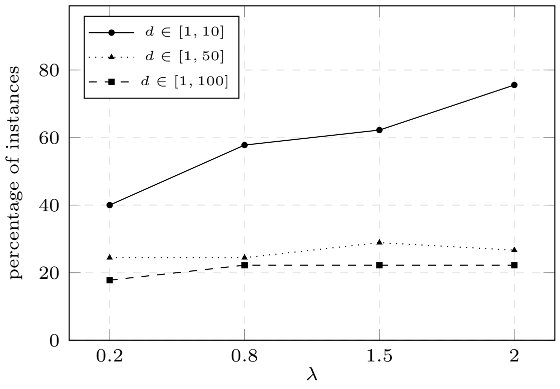
Table 3: Performance of CPLEX and the heuristic approach for the small test instances

Instance set			CPLEX			Heuristic			
s	k	n	feas. [%]	opt. [%]	t_{avg} [s]	gap_{avg}^{feas}	gap_{avg}^{opt}	t_{avg} [ms]	t_{max} [ms]
5	2	10	100	100	35.85	6.41	6.41	0.07	0.18
		20	100	75	360.8	-3.11	3.25	0.08	0.12
		30	100	25	458.09	-25.01	0	0.17	0.67
		40	100	33.33	56.75	-25.46	2.19	0.17	0.26
		50	91.67	8.33	160.65	-29.24	0	0.32	1.04
7	2	14	100	100	257.93	2.57	2.57	0.13	0.39
		28	100	33.33	45.96	-12.59	1.89	0.17	0.24
		42	100	16.67	20.97	-24.93	10.47	0.26	0.38
		56	91.67	16.67	348.19	-26.41	0.67	0.57	1.83
		70	75	8.33	117.22	-37.44	0	0.58	1.91
7	3	14	100	91.67	14.56	6.03	6.17	0.09	0.15
		28	100	50	13.26	-12.19	1.04	0.3	0.6
		42	100	25	13.88	-22.37	0.48	0.22	0.32
		56	100	16.67	15.6	-27.91	1.32	0.29	0.39
		70	100	16.67	405.5	-21.62	0.52	0.32	0.39
9	2	18	100	66.67	118.75	-2.95	5.89	0.15	0.2
		36	100	33.33	156.82	-20.61	4.89	0.43	1.32
		54	91.67	33.33	298.51	-25.28	5.21	0.47	0.65
		72	91.67	25	181.93	-27.25	5.09	0.63	0.86
		90	91.67	0	-	-38.45	-	1.2	3.07
9	3	18	100	83.33	62.37	0.58	0.59	0.14	0.41
		36	100	33.33	50.21	-20.37	0.98	0.61	1.32
		54	100	16.67	181.87	-23.34	0.56	1.63	1.92
		72	100	8.33	295.87	-22.75	19.47	1.72	2.33
		90	91.67	16.67	73.1	-32.8	0	2.29	3.78
9	4	18	100	100	21.7	2.85	2.85	0.1	0.15
		36	100	50	12.3	-15.66	6.53	0.37	1.02
		54	100	25	229.26	-18.72	9.85	1.15	1.72
		72	100	16.67	522.99	-18.83	0.69	1.32	1.94
		90	75	8.33	295.09	-25.65	0	1.98	2.67
11	2	22	100	50	152.06	-10.1	3.65	0.43	1.02
		44	100	33.33	196.23	-24.65	4.63	0.85	2.05
		66	91.67	8.33	113.59	-33.89	0	0.85	2.78
		88	91.67	8.33	20.95	-37.19	0	0.95	1.61
		110	83.33	0	-	-41.32	-	1.27	1.82
11	3	22	100	83.33	91.81	-7.35	0	0.18	0.27
		44	100	41.67	47.29	-13.82	3.91	0.37	0.61
		66	100	25	431.11	-24.24	7.64	0.53	0.69
		88	100	8.33	12.87	-24.99	0	0.65	0.81
		110	66.67	0	-	-36.72	-	0.9	1.14
11	5	22	100	100	51.87	2.8	2.8	0.14	0.22
		44	100	41.67	95.77	-8.13	3.78	0.28	0.43
		66	100	33.33	17.58	-13.5	4.87	0.38	0.63
		88	83.33	16.67	454.32	-18.7	2.78	0.5	0.65
		110	83.33	8.33	57.88	-33.51	0	0.65	1.24

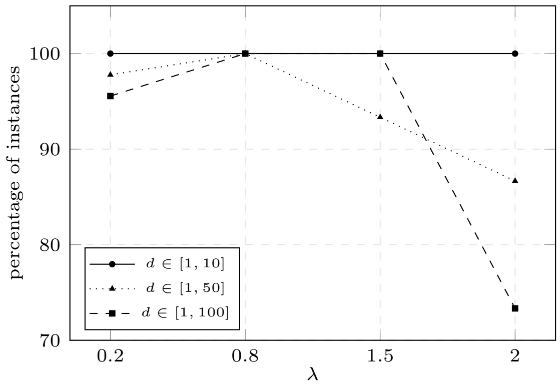
heuristic approach, it presents the average gap when solely considering all instances where CPLEX found feasible (column "gap_{avg}^{feas}") or optimal (column "gap_{avg}^{opt}") solutions, as well as the average (column "t_{avg}") and maximum (column "t_{max}") runtime in milliseconds.

As to be expected, an increase of the number of drones s or jobs n tends to result in CPLEX performing worse in terms of the number of instances where a feasible or optimal solution is found. On the other hand, CPLEX tends to benefit from an increasing capacity k. Our heuristic approach terminated with a feasible solution for all instances. Its maximum runtimes are in the range of only a few milliseconds. The average gaps show that the heuristic approach is competitive when compared with CPLEX. When considering all instances where CPLEX determined feasible solutions, the results indicate that the heuristic (on average) outperforms CPLEX for $n \geq 4s$. Thus, we can conclude that our straightforward solution procedure performs surprisingly well in terms of runtime and solution quality. It is therefore certainly an adequate method for deriving managerial insights, which is our main intention in the next section.

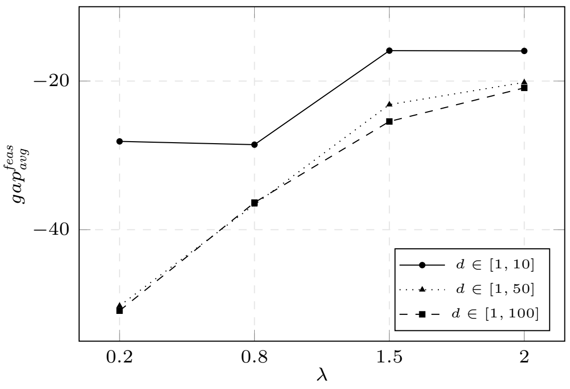
To analyze the influence of larger processing times and increasing values of the travel time d from the depot to the launching point, we grouped the instances based on the values of λ and d given in Table 2. Figure 4 illustrates the related computational results in three plots for the instances where CPLEX determined optimal (Figure 4a) or feasible (Figure 4b) solutions, and for the average gap of our heuristic approach (Figure 4c). As



(a) CPLEX: optimal solution determined



(b) CPLEX: feasible solution determined



(c) Heuristic: average gap

Figure 4: Impact of larger processing times and increasing values of d

can be seen, CPLEX benefits from smaller values of d when considering its ability to find optimal solutions. A potential reason of this effect is the fact that the upper bound (14) increases with lager values of d. Interestingly, smaller average processing times (caused by an increasing λ), while reducing the value of (14), induce a similar effect. The ability of CPLEX to determine feasible solutions is best for medium average processing times. The average gap of the solutions determined by our heuristic approach is in line with the behavior of CPLEX. Again, we find that our approach seems well suited for drawing managerial conclusions in the next section.

6.3. Managerial Implications

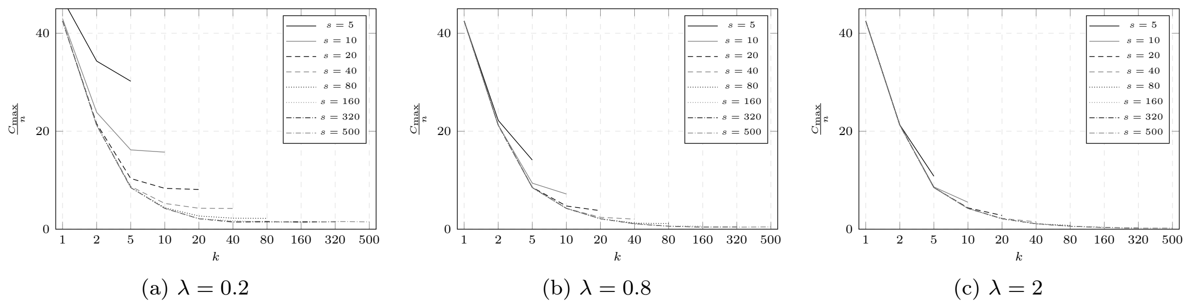
The main cost drivers when setting up a piggyback transportation process are certainly the capacity k of the large vehicle and the fleet size s of small vehicles. We want to explore where investment is better spent and how the answer to this question is impacted by the population density (steered by parameter λ). To address these issues, we generated additional random test instances with a larger number of jobs n and drones s. The instances are such that we are able to analyze the effects of doubling the number of drones or the divisor used to compute the capacity in different settings. Table 4 lists the parameters used to generate the testbed. Here, $\bar{s}$ and

Table 4: Parameter values of managerial test instances

n	$\bar{s}$	$\bar{k}$	d	λ
100, 500, 1000	$\{5 \cdot 2^y \mid y = 0, \ldots, 8\}$	$\{\lfloor \frac{\bar{s}}{2^y} \rfloor \mid y = 0, \ldots, 10\}$	5, 10, 20, 50	0.2, 0.8, 2

$\bar{k}$ are auxiliary parameters applied to derive the number of drones s and the capacity k of the shuttle. For each combination of the parameters presented in the table, we randomly generated five instances with $s = \min\{\bar{s}, n\}$ and $k = \min\{\bar{k}, s\}$ as described in Section 6.2. We discarded all instances with $k < 1$ and all but five instances for each resulting combination of n, s, k, d, and λ. This results in a total of 8880 instances. Our heuristic approach returned a feasible solution for all of these instances. The average runtimes (again, the overall runtime for a single instance includes four runs of Algorithm 2) were in the range of 4 millisecond for instances with $n = 100$, 70 milliseconds for instances with $n = 500$, and 0.3 seconds for instances with $n = 1000$. The maximum runtime was around 8 seconds for an instance with $n = 1000$, $s = 320$, $k = 1$, $d = 5$, and $\lambda = 0.8$.

We first analyze the effect of increasing expected average processing times for given values of d. For the sake of brevity, we restrict ourselves to the case $n = 500$. Figures 5 and 6 illustrate the corresponding computational results. Essentially, both figures include the same information, namely the effect of varying s and k on the

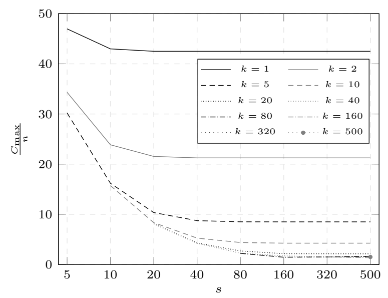


(a) $\lambda = 0.2$ (b) $\lambda = 0.8$ (c) $\lambda = 2$

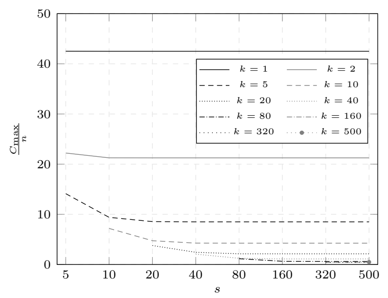
Figure 5: Impact of increasing capacity k for different values of λ with $n = 500$

average value of $C_{\max}/n$ in the corresponding set of instances. However, in order to highlight the related effects more clearly, Figure 5 focuses on increasing values of k, while Figure 6 displays the impact of increasing values of s. The results of Figures 5 and 6 suggest the following findings:

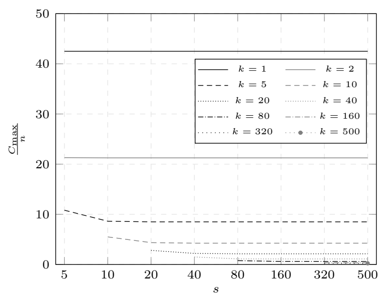
- *Impact of capacity k:* Naturally, increasing the large vehicle's capacity k beyond the fleet size s of small vehicles makes no sense. Inevitably, the extra capacity remains unused. However, even before reaching this upper limit, we can observe that the positive effect of increasing capacity k quickly diminishes. This effect is even more distinct, if we have small λ values. In this case, the variation of processing times p_j



(a) $\lambda = 0.2$



(b) $\lambda = 0.8$



(c) $\lambda = 2$

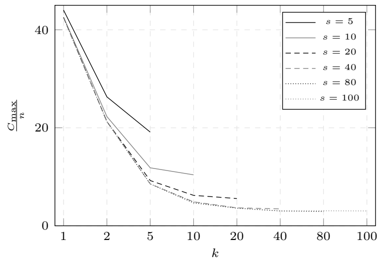
Figure 6: Impact of increasing number s of drones for different values of λ with $n = 500$

is large, and we have a fairly large probability that some customers residing in the outskirts require large processing times. Thus, we have a larger probability that drones servicing these far-away customers have not yet returned when the shuttle arrives at the depot from the respective round trip, so that extra capacity remains unused. If λ is large, instead, we have the case of a crowded city center, where all customers require a processing time p_j close to the return time d of the shuttle; larger processing times to outskirt customers are rather unlikely. In this case, extra capacity is much less likely to remain unused. However, even in this case, increasing capacity k beyond a few dozen drones does not (considerably) reduce the makespan in our experiments, because, in any case, we have to wait for the return of the drones with the farthest flight legs, which are released on the early round trips of the large vehicle.

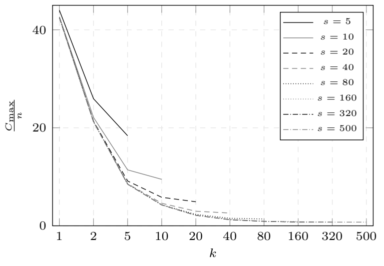
- *Impact of fleet size s:* The diminishing return of an increasing fleet size s of small vehicles is even more pronounced. For large λ, i.e., a densely populated city center where the drones have to service only few outskirt customers with long flight times, increasing the fleet size of drones beyond capacity k makes not much sense. Here, most processing times are close to d, so that, in most cases, enough drones have returned to the depot to instantaneously load the shuttle to full capacity for its next round trip. The probability of drones having not yet returned to the depot increases for smaller values of λ. In this case, having a few more drones than shuttle capacity makes more sense, because extra drones can fill the gaps of those having not yet returned. However, even with many outskirt customers, having more than twice as many drones than capacity k barely improves the makespan in our experiments.

- These results can be translated into the following simple *take-home messages:* Invest into a sufficiently large shuttle capacity k. This capacity should be well below the value $\frac{n}{\frac{p_{max}}{D}}$, where p_{max} refers to the maximum processing time over all jobs. Dividing the maximum processing time by the time D needed for a single round trip of the shuttle, results in the number of round trips that can be executed while the farthest customer is serviced. Thus, dividing the customer base n, typically to be serviced during a work shift, by this number, delivers an upper limit for a reasonable capacity k. If the service region is limited to a densely populated city center, then the drone fleet size s should equal capacity k plus a handful of extra drones. Only if a lot of outskirt customers with considerably larger processing times occur regularly, increasing the drone fleet size up to $2k$ can be reasonable.

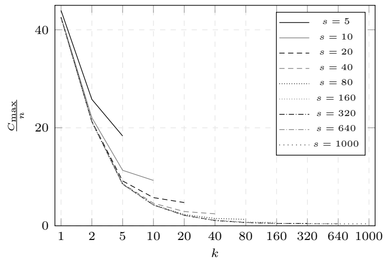
Note, however, that our experiments abstract from investment and operational costs on the one hand and the benefits of a better service quality and faster deliveries on the other. Additional experiments including these values should be executed if a specific piggyback process is to be established. Further impact is to be expected by the size n of the customer base to be serviced. To analyze the effect of increasing values of n, we grouped the instances in sets with identical values of s, k, and n. Figures 7 and 8 illustrate the corresponding computational results. Generally, these results confirm our previous findings. Not that obvious, however, is the fact that more



(a) $n = 100$

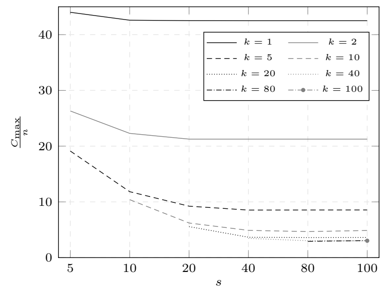


(b) $n = 500$

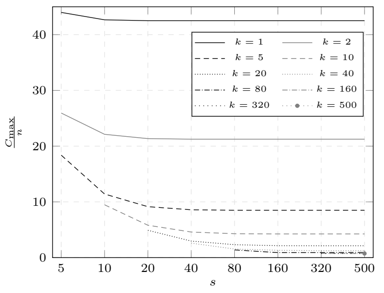


(c) $n = 1000$

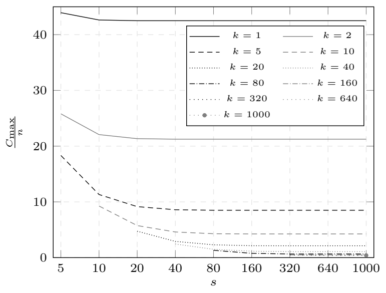
Figure 7: Impact of increasing capacity k for different numbers n of customers



(a) $n = 100$



(b) $n = 500$



(c) $n = 1000$

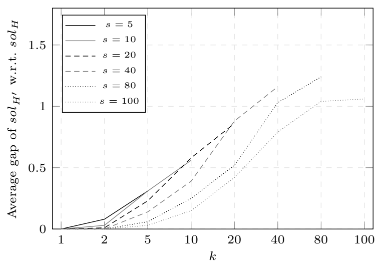
Figure 8: Impact of increasing number of drones s for different numbers n of customers

customers n to be serviced tend to slightly increase the positive impact of more capacity k and drones s than for smaller n. The reason for this is that the larger the customer base to be serviced, the larger the probability that suited customers fitting into a shuttle schedule without delay exist. Thus, we can add the following extension to our take-home messages: When testing a piggyback process such as the flying warehouse where the customer base is still small, diminishing returns of additional capacity and drones are especially pronounced. Thus, in early phases, investment costs can be kept low. With a successful diffusion of piggyback services and an increasing customer base, larger shuttles and drone fleets are a much better (and safer) invest.

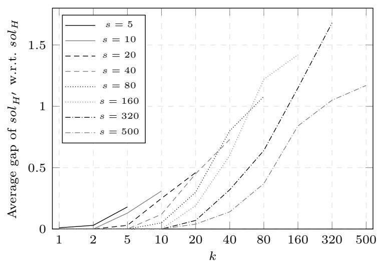
We close this section with an analysis of the effect of balancing the workload among drones within the basic heuristic approach introduced in Section 6.1. As mentioned above, we apply the procedure Best-Fit (Algorithm 1) to determine well balanced drone schedules. To analyze the effect of using this approach, we compare it to the results of Algorithm 2 when fed with random drone schedules, i.e., schedules that are generated by iterating over the jobs, selecting a random drone for each job (each with identical probability), and appending the job at the end of the drone schedule. For some instance I of PTP, we define $sol_{H'}$ to be the best solution determined by the four runs of Algorithm 2 on these modified drone schedules. Furthermore, in line with the definition of (15), we define the gap of $sol_{H'}$ with respect to sol_H as

$$\frac{C_{\max}(sol_{H'}) - C_{\max}(sol_H)}{C_{\max}(sol_H)}.$$

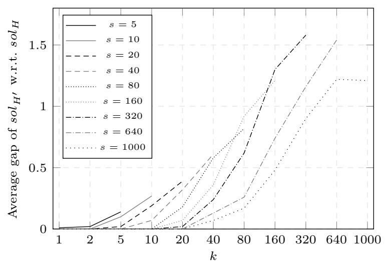
Figure 9 displays the average gaps in our computational experiments for the different values of n and s over the shuttle capacity. As can be seen, the average gaps range to values of around 1.5, so that there clearly is a major negative impact that results from neglecting workload balancing by using random drone schedules. For relatively large shuttle capacities and few drones, well balanced drone schedules become increasingly important.



(a) $n = 100$



(b) $n = 500$



(c) $n = 1000$

Figure 9: Effect of balancing drone workload for different numbers n of customers

This, of course, must be interpreted in light of the fact that our algorithmic approach separately generates drone schedules before computing round trips. In general, however, we can conclude that one must specifically address the synchronization property when developing heuristics for PTP.

7. Conclusions

This paper addresses a very basic transportation problem, called the piggyback transportation problem: A large vehicle takes a number of small vehicles on board and transports them toward an advantageous launching point, where they are launched with shipments toward customer homes. Then, the large vehicle returns back to the depot, which is the collection point for all returning vehicles, and takes another batch of small vehicles on board until all customers are serviced and all vehicles have returned to the depot. This problem setting is, for instance, relevant for Amazon's flying warehouse concept, where a shuttle delivers drones toward a flying warehouse hovering over a city center. We prove several important properties of our piggyback transportation problem, such as the complexity status, efficient solution algorithms for restricted problem settings, and a polynomial-time approximation scheme for the problem. Based on these structural insights, we provide a simple, yet very efficient heuristic solution procedure. This procedure is finally applied to explore how an investment into a piggyback transportation process should be split among the loading capacity of the large vehicle and the fleet size of small vehicles, including the prioritization of delivery capacity (see Chen et al. 2021). The good news for practitioners having to set up a piggyback transportation process is that the positive impact of additional capacity and larger fleet sizes quickly diminish, so that investment costs can be kept low without deteriorating service quality.

We see several avenues for future research (also for the operational research community) to finally bring piggyback processes such as the flying warehouse to a market-ready state. For instance, in our problem setting, the launching position remains stable over the innermost city center. Following the expectations of Poikonen & Golden (2020), however, the flying warehouse could also change its position during the day to improve the launching positions toward other urban districts. Over time, this would impact the flight times d of the shuttle and the processing times for the drone flights in our problem setting. Also, the limited flight ranges of the drones, which may not be able to reach any customer from any launching point, need to be considered in this case. The resulting holistic optimization problem also has to determine the flight path of the warehouse and coordinate it with the piggyback delivery process. For such an extended problem setting and also our core problem, other objective functions could be applied to measure service quality. Alternative (e.g., due date related) objectives and additional constraints (e.g., an additional resource successively loading shipments onto small vehicles at the launching point) lead to interesting variations of our basic piggyback transportation problem, for which important structural properties are yet unknown.

Acknowledgements

This work has been supported by the German Research Foundation (DFG) through the grant "Sustainable Personnel Planning in Highly Customized Assembly Lines with Work Sharing" (KR 4926/3-1, OT 500/4-1). Kai Wang and Ilia Fridman have been supported by the Damp and the Friede Springer foundation (grant no. 60 0 0108), respectively.

References

Agatz, N., Bouman, P., & Schmidt, M. (2018). Optimization approaches for the traveling salesman problem with drone. *Transportation Science*, *52*, 965–981.

Berg, P. W., Isaacs, S., & Blodgett, K. L. (2016). U.S. Patent No. 9,305,280. Washington, DC: U.S. Patent and Trademark Office.

Blazewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., Sterna, M., & Weglarz, J. (2019). *Handbook on Scheduling: From Theory to Practice*. (2nd ed.). Berlin: Springer.

Boysen, N., Schwerdfeger, S., & Weidinger, F. (2018). Scheduling last-mile deliveries with truck-based autonomous robots. *European Journal of Operational Research*, *271*, 1085–1099.

Brucker, P., Gladky, A., Hoogeveen, H., Kovalyov, M. Y., Potts, C. N., Tautenhahn, T., & Van De Velde, S. L. (1998). Scheduling a batching machine. *Journal of Scheduling*, *1*, 31–54.

Chen, H., Hu, Z., & Solak, S. (2021). Improved delivery policies for future drone-based delivery systems. *European Journal of Operational Research*, *https://doi.org/10.1016/j.ejor.2021.02.039*.

Chen, L., Jansen, K., & Zhang, G. (2014). On the optimality of approximation schemes for the classical scheduling problem. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms* (pp. 657–668).

Cordeau, J.-F., Laporte, G., Moccia, L., & Sorrentino, G. (2011). Optimizing yard assignment in an automotive transshipment terminal. *European Journal of Operational Research*, *215*, 149–160.

Garey, M. R., & Johnson, D. S. (1978). "Strong" NP-completeness results: Motivation, examples, and implications. *Journal of the ACM (JACM)*, *25*, 499–508.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability - A Guide to the Theory of NP-Completeness*. New York: Freeman.

Graham, R. L. (1969). Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, *17*, 416–429.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In P. L. Hammer, E. L. Johnson, & B. H. Korte (Eds.), *Annals of Discrete Mathematics, Vol. 5* (pp. 287–326). Amsterdam: North-Holland.

Hochbaum, D. S., & Shmoys, D. B. (1987). Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, *34*, 144–162.

Hulett, H., Will, T. G., & Woeginger, G. J. (2008). Multigraph realizations of degree sequences: Maximization is easy, minimization is hard. *Operations Research Letters*, *36*, 594–596.

Jia, Z.-h., & Leung, J. Y.-T. (2015). A meta-heuristic to minimize makespan for parallel batch machines with arbitrary job sizes. *European Journal of Operational Research*, *240*, 649–665.

Karp, R. M. (1972). Reducibility among combinatorial problems. In R. E. Miller, & J. W. Thatcher (Eds.), *Complexity of Computer Computations* (pp. 85–104). New York: Plenum Press.

Liu, M., Chu, F., He, J., Yang, D., & Chu, C. (2019). Coke production scheduling problem: A parallel machine scheduling with batch preprocessings and location-dependent processing times. *Computers & Operations Research*, *104*, 37–48.

Malapert, A., Guéret, C., & Rousseau, L.-M. (2012). A constraint programming approach for a batch processing problem with non-identical job sizes. *European Journal of Operational Research*, *221*, 533–545.

Mattfeld, D. C., & Branke, J. (2005). Task scheduling under gang constraints. In G. Kendall, E. Burke, S. Petrovic, & M. Gendreau (Eds.), *Multidisciplinary Scheduling: Theory and Applications* (pp. 113–130). New York: Springer.

Mattfeld, D. C., & Kopfer, H. (2003). Terminal operations management in vehicle transshipment. *Transportation Research Part A: Policy and Practice*, *37*, 435–452.

Mattfeld, D. C., & Orth, H. (2006). The allocation of storage space for transshipment in vehicle distribution. *OR Spectrum*, *28*, 681–703.

Otto, A., Agatz, N., Campbell, J., Golden, B., & Pesch, E. (2018). Optimization approaches for civil applications of unmanned aerial vehicles (UAVs) or aerial drones: A survey. *Networks*, *72*, 411–458.

Poikonen, S., & Golden, B. (2020). The mothership and drone routing problem. *INFORMS Journal on Computing*, *32*, 249–262.

Thoung, L. T. (1989). From piggyback to double-stack intermodalism. *Maritime Policy and Management*, *16*, 69–81.

Wang, C.-S., & Uzsoy, R. (2002). A genetic algorithm to minimize maximum lateness on a batch processing machine. *Computers & Operations Research*, *29*, 1621–1640.