This is a preliminary version of the published paper:

Alessandro Agnetis, Matteo Cosmi, Gaia Nicosia, Andrea Pacifici,
Two is better than one? Order aggregation in a meal delivery scheduling problem,
Computers & Industrial Engineering, Volume 183, 2023, 109514,
ISSN 0360-8352, https://doi.org/10.1016/j.cie.2023.109514. (https://www.sciencedirect.com/science/article/pii/S0360835223005387)

An order aggregation and scheduling problem for meal delivery

Alessandro Agnetis* Matteo Cosmi† Gaia Nicosia† Andrea Pacifici‡

May 14, 2021

Abstract

We address a single-machine scheduling problem motivated by a last-mile-delivery setting for a food company. Customers place orders, each characterized by a delivery point (customer location) and an ideal delivery time. An order is considered on time if it is delivered to the customer within a time window given by the ideal delivery time $\pm\delta$, where δ is the same for all orders. A single courier (machine) is in charge of delivery to all customers. Orders are either delivered individually, or two orders can be aggregated in a single courier trip. All trips start and end at the restaurant, so no routing decisions are needed. The problem is to schedule courier trips so that the number of late orders is minimum. We show that the problem with order aggregation is NP-hard and propose a combinatorial branch and bound algorithm for its solution. The algorithm performance is assessed through a computational study on instances derived by a real-life application and on randomly generated instances. The behavior of the combinatorial algorithm is compared with that of the best ILP formulation known for the problem. Through another set of computational experiments, we also show that an appropriate choice of design parameters allows applying the algorithm to a dynamic context, with orders arriving over time.

Keywords: Scheduling; Complexity; Integer programming; Branch and bound; Food delivery; Last-mile delivery.

1 Introduction

In this paper we address a scheduling problem called *single-courier meal delivery scheduling problem* (*MDSP*). In fact, it is motivated by a food pick-up and delivery application in

*Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche, Università degli Studi di Siena, Via Roma 56, 53100 Siena, Italy. agnetis@diism.unisi.it

†Dipartimento di Ingegneria, Università degli studi Roma Tre, Via della Vasca Navale 79, 00146 Roma, Italy. {matteo.cosmi, gaia.nicosia}@uniroma3.it

‡Dipartimento di Ingegneria Civile e Ingegneria Informatica, Università degli Studi di Roma "Tor Vergata", via del Politecnico 1, 00133 Roma, Italy. andrea.pacifici@uniroma2.it

Rome, Italy. A restaurant collects food orders from the final customers. The customers specify their location and the ideal delivery time (besides, of course, the type and amount of required dishes). Meals are delivered by a single courier. The problem is to schedule the courier's trips so that the number of orders which are not delivered within their time window is minimized. This is a sensible objective, since one can assume that missed orders are outsourced to a third-party logistics operator charging a fixed rate for each order. We assume that the restaurant is non-bottleneck, i.e., meals are always ready when needed for delivery.

During each roundtrip, the courier can deliver a limited number of orders. In fact, to cope with the congested traffic of big cities, couriers typically move around by bicycles or mopeds, and hence can carry no more than *one or two* orders. (This is a relevant difference from, e.g., parcel delivery which is typically carried out through vans or small trucks.) In particular, we let $MDSP_S$ denote the problem in which all trips carry a *single order*, and $MDSP_A$ the problem in which up to two orders can be *aggregated* in the same trip.

A relevant feature of our meal delivery problem is that each delivery should take place in a time window of width δ, centered around the ideal delivery time specified by the customer. The value of δ is the same for all orders, and it expresses service timeliness. The consequence (as shown in detail in Section 2) is that, if we consider the problem in which each trip delivers a single order ($MDSP_S$), our problem reduces to a single-machine scheduling problem in which, denoting by r_j, d_j and p_j release date, due date and processing time of a job j, it turns out that for all j

$$d_j - r_j - p_j = \delta,$$

i.e., all jobs have the same *slack*. Scheduling problems with fixed or limited slack have been addressed in the literature, as reviewed in Section 2.

In this paper we consider the more general case in which the courier is allowed to serve up to two different orders in a single trip, that we refer to as *order aggregation* (problem $MDSP_A$). This additional feature results in a substantially different scheduling model. Such a variant of the problem has been introduced by Cosmi et al. (2019a), where different lower bounds are compared. In Cosmi et al. (2019b), computational results are presented concerning a number of integer optimization models for the problem.

$MDSP$ is conceptually similar to the so-called *same-day delivery problem* ($SDDP$). Also in this case, consumers place orders for the same day, and the orders are fulfilled drawing items from a centralized inventory. However, $SDDP$ is stochastic and dynamic (see Ulmer et al. (2020)), that is, online orders are not known a priori and are only revealed over time.

$SDDP$ is indeed strongly related to the vehicle routing problem with release dates (Archetti et al. (2015), Azi et al. (2012)).

Among the most recent literature on $SDDP$, the following works specifically deal with deliveries performed by a single vehicle. In Ulmer et al. (2019) the problem with a single vehicle and stochastic order arrival is addressed. The vehicle is allowed to (preemptively) return to the depot before delivering all loaded packages. To solve the problem, they combine an approximation procedure based on dynamic programming that chooses the subset of requests for delivery with a routing heuristic. Klapp et al. (2018b) consider a problem with a single vehicle and orders distributed on a line. The decision process is divided in epochs characterized each by a set of known delivery requests and a set of potential requests. At each epoch the goal is to decide whether or not to dispatch the vehicle, loaded with known orders, so that expected operational costs and penalties for unserved orders are minimized. In Klapp et al. (2018a), the same authors assume a more general network topology and provide a more realistic model of the same-day delivery operations in a typical road network.

Different from $SDDP$, it is important to stress that in $MDSP_A$ deliveries contain at most two orders, and delivery-time requirements are stricter than in most of the above models. Hence, in $MDSP_A$ scheduling decisions, rather than routing decisions, are crucial.

The paper is organized as follows. In Section 2 we formally describe $MDSP_S$, and we define the main notation used in the paper. In Section 3 we introduce the concept of order aggregation and the corresponding additional notation, give a formal definition of problem $MDSP_A$ and characterize its complexity. In Section 4 some lower bounds are described, exploited in a branch and bound algorithm presented in Section 5. Section 6 presents the results of an extensive computational campaign. Finally, in Section 7 some conclusions are drawn.

2 The meal delivery problem with single-order trips ($MDSP_S$)

In this section we introduce notation and establish some properties of our problem in which, during each courier trip, a single order is delivered, so in this case there is a one-to-one correspondence between orders and courier's trips.

A set of *orders* $J = \{1, \ldots, n\}$ is given. (Since each order corresponds to a customer, we indifferently use the terms *order* and *customer*.) For each order $j \in J$, an *ideal delivery time* $\hat{d}_j$ and a restaurant-to-destination *travel time* t_j are specified. The trip corresponding to order j consists in the courier leaving the restaurant, reaching customer j, and heading back to the restaurant.

The *delivery time* of an order j equals the courier *start time* at the restaurant plus the *travel time* between the restaurant and destination j. An order is considered on time if it is delivered in an interval of δ minutes centered around the ideal delivery time chosen by the customer. As long as the courier is traveling, she is not available for processing any other order until she is back again to the restaurant. So, in $MDSP_S$ we may view the orders as *jobs* and the courier as a processing resource, and each job $j \in J$ is associated with the following data. The *due date* d_j is the latest possible time for the courier to be back at the restaurant after delivering order j on time to the customer, i.e.,

$$d_j = \hat{d}_j + \frac{1}{2}\delta + t_j. \tag{1}$$

The *release date* r_j is the earliest possible time for the courier to start from the restaurant and deliver order j on time to the customer, i.e.,

$$r_j = \hat{d}_j - \frac{1}{2}\delta - t_j \tag{2}$$

The *processing time* p_j equals the amount of time the courier is busy with order j, i.e., the total roundtrip time

$$p_j = 2t_j. \tag{3}$$

(We implicitly assume that loading/unloading times are included in travel times.)

In this framework, an order j is on time if and only if the corresponding job starts (i.e., the courier picks up the food at the restaurant) not before r_j and completes (i.e., the courier returns to the restaurant) not later than d_j. An order which is not on time is *tardy*. As a consequence of the above definitions, one has:

$$d_j = r_j + p_j + \delta \tag{4}$$

which makes our scheduling problem a special case of problem $1|r_j|\sum U_j$, in which due dates and release dates are interdependent, namely, while due dates and processing times may vary, the difference $d_j - r_j - p_j$ is the same (equal to δ) for all j. This quantity is known as *slack* or *additive laxity* (Böhm et al. (2021)).

While problem $1|r_j|\sum U_j$ is, in general, strongly $\mathcal{NP}$-hard (Garey and Johnson (1979)), a number of papers addressed the case in which constraints exist on slack. For the case in which the slack *must not exceed* a certain value δ, Cieliebak et al. (2004) address the problem of minimizing the number of machines necessary to complete all jobs on time. The authors show that their problem can be solved in polynomial time if $\delta \in \{0, 1\}$. Otherwise, they propose a solution algorithm that runs in $O(n(\delta+1)^H H \log H)$, where n and H are the number of jobs and the maximum number of overlapping job time windows, respectively. A refined algorithm for the *feasibility* version of the same problem is proposed in van Bevern et al. (2017). Its complexity is $O(n\delta m \log(\delta m)(\delta+1)^{m(2\delta+1)} + n \log n)$, where m is the number of available machines. While it is shown in Cieliebak et al. (2004) that this problem is already $\mathcal{NP}$-hard when $\delta = 2$ for arbitrary m, it is tractable for fixed parameter $m + \delta$ (van Bevern et al. (2017)). Cosmi et al. (2019c) show that the optimization version of the same problem with $m = 1$ can be solved in $O(n(\delta + 2)^{2(\delta+1)} + n \log n)$. When $m = 1$ and (4) holds, which is the case of our MDSP without order aggregation, Böhm et al. (2021) recently proposed an algorithm that exactly solves this problem in $O(n^2)$. So, $MDSP_S$ is indeed polynomially solvable. We next see that this is not the case when order aggregation is allowed ($MDSP_A$).

3 The meal delivery problem with order aggregation ($MDSP_A$)

In this section, we consider $MDSP_A$, i.e., the problem in which a courier may pick up either a single order or a *pair* of orders to be delivered in a single trip. In the latter case, the deliveries to the two customers are sequentially performed in the same trip from the restaurant to the two different locations and back. In the following we refer to such a composite trip as a *twin*. When performing a twin, we suppose that after the first delivery, the courier immediately proceeds to deliver the second order, i.e., we assume that the following condition holds.

No-wait assumption: While performing a twin, the courier is not allowed to introduce idle time between the two deliveries.

This assumption comes from the application scenario motivating this study. In fact, such a policy prevents decays in the quality of the delivered food. Moreover, one typically wants that the courier spends traveling only the time strictly necessary to reach the customers.

Clearly, a courier may deliver two orders i and j in a single trip (and hence, orders i and j are allowed to be in the same twin) only if it is possible to meet the corresponding delivery-time constraints under the no-wait assumption. In this case, the twin composed by order i followed by order j (hereafter, for notation simplicity, indicated by ij) is called a *feasible twin.* In the following, we let t_{ij} denote the travel time from customer i to customer j.

Similar to $MDSP_S$, taking into account travel times from/to the restaurant and between two customers, we can define processing times, release dates, and due dates for each twin ij. The *processing time* p_{ij} represents the total amount of time the courier is busy with the twin ij. Due to the no-wait assumption, j is reached $t_i + t_{ij}$ time units after the courier starts from the restaurant, while the same courier is back after p_{ij} time units, where

$$p_{ij} \stackrel{\text{def}}{=} t_i + t_{ij} + t_j. \tag{5}$$

Recalling that $\hat{d}_k - \delta/2$ is the earliest time at which customer k can be reached, the *release date* r_{ij} of the twin ij is the earliest possible time for the courier to start from the restaurant and deliver both orders i and j on time, i.e.,

$$r_{ij} \stackrel{\text{def}}{=} \max\left\{\hat{d}_i - \frac{\delta}{2} - t_i, \hat{d}_j - \frac{\delta}{2} - (t_i + t_{ij})\right\}. \tag{6}$$

Since $\hat{d}_k + \delta/2$ is the latest time at which customer k can be reached, the *due date* d_{ij} of the twin ij is the latest possible time for the courier to be back to the restaurant after delivering both the orders i and j on time, i.e.,

$$d_{ij} \stackrel{\text{def}}{=} \min\left\{\hat{d}_i + \frac{\delta}{2} + t_{ij} + t_j, \hat{d}_j + \frac{\delta}{2} + t_j\right\}. \tag{7}$$

Hence, a pair of customer orders i and j may constitute a feasible twin if a courier starting not before r_{ij} is able to deliver the two orders and return back to the restaurant within the due date d_{ij}.

Proposition 1. *Given two orders $i, j \in J \times J$, the twin ij is feasible if and only if:*

$$\delta_{ij} \stackrel{def}{=} \delta - |t_{ij} - d_j + d_i + 1/2(p_j - p_i)| \geq 0 \tag{8}$$

Proof. Recall that, for any order $j \in J$, the processing time p_j is twice the travel time t_j from the restaurant to customer j and that (4) holds. In (6), we notice that, by (2), the first term in the max expression $\hat{d}_i - \frac{\delta}{2} - t_i$ equals r_i. Since $\hat{d}_j - \frac{\delta}{2} = r_j + t_j$, the second term can be rewritten as $r_j + \frac{1}{2}(p_j - p_i) - t_{ij}$.

Similarly, the second term of the min function in (7), by (1), is equal to d_j. The first term, since $\hat{d}_i + \frac{\delta}{2} = d_i - t_i$, is equal to $d_i + \frac{1}{2}(p_j - p_i) + t_{ij}$.

Now, let $\tau_{ij} \stackrel{def}{=} t_{ij} - (d_j - d_i) + \frac{1}{2}(p_j - p_i)$. Due to (1), it is immediate to verify that the following expression holds:

$$\tau_{ij} \geq 0 \Leftrightarrow t_{ij} \geq \hat{d}_j - \hat{d}_i. \tag{9}$$

As a consequence, we can rewrite (6) and (7) as:

$$r_{ij} = \begin{cases} r_i & \text{if } \tau_{ij} \geq 0; \\ r_j + \frac{1}{2}(p_j - p_i) - t_{ij} & \text{otherwise.} \end{cases} \tag{10}$$

$$d_{ij} = \begin{cases} d_j & \text{if } \tau_{ij} \geq 0; \\ d_i + \frac{1}{2}(p_j - p_i) + t_{ij} & \text{otherwise.} \end{cases} \tag{11}$$

The above relations show that, if the travel time between the two customers is larger than the difference between their ideal delivery times, then the release date (resp. due date) of the twin is equal to the release date of the first order i (resp. the due date of the second order j). Otherwise, the other terms of expressions (10) and (11) dominate.

Clearly, in order for twin ij to be feasible, one must have:

$$r_{ij} + t_i + t_{ij} + t_j \leq d_{ij}.$$

If $\tau_{ij} \geq 0$, recalling that $r_i = d_i - p_i - \delta$, this condition can be rewritten as $\delta - \tau_{ij} \geq 0$. Viceversa, if $\tau_{ij} \leq 0$, with some algebra, one can show that the same condition is equivalent to $\delta + \tau_{ij} \geq 0$. In conclusion, (8) can be written as

$$\delta_{ij} = \delta - |\tau_{ij}| \geq 0$$

i.e., ij is a feasible twin if and only if $\delta_{ij} \geq 0$. □

Note that, similar to δ for a standard (single) trip, δ_{ij} plays the role of a slack time for a feasible twin. In fact, it is easy to see that for any twin ij, it holds

$$d_{ij} = r_{ij} + p_{ij} + \delta_{ij}.$$

However, note that $\delta_{ij} \leq \delta$ and in general it depends on the pair (i, j) of orders forming the twin.

We denote the set of feasible twins by

$$D = \{ij \ : \ (i,j) \in J \times J, \ \delta_{ij} \geq 0\}. \tag{12}$$

For notational convenience, in D we also include the special symbol jj (for any $j \in J$) to represent a single order j (not aggregated to another order) as a twin. Clearly, $jj \in D$ for all $j \in J$, since, from (8), $\delta_{jj} = \delta$. Note that $\tau_{jj} = 0$ and hence $r_{jj} = r_j$ and $d_{jj} = d_j$. Hence, from now on we do not distinguish between trips containing one or two orders, regarding all trips as twins.

Naturally extending the concept of on-time order, we say that a twin $ij \in D$ *is on time* if both its component orders are so. Notice that the feasibility condition (8) is only necessary for a twin ij to be on time. Indeed, ij is on time if and only if its *starting time* s_{ij} belongs to the interval $[r_{ij}, r_{ij} + \delta_{ij}]$.

We can now formally define the problem addressed in the remainder of the paper.

> Problem $MDSP_A$
> **Given:** n customer orders $J = \{1,2,\ldots,n\}$, the ideal delivery time $\hat{d}_j$, $j = 1,\ldots,n$, the travel time t_j between the restaurant and customer j, $j = 1,\ldots,n$, the travel time t_{ij} between customers i and j, $i,j = 1,\ldots,n$ ($t_{jj} = 0$), a slack value $\delta > 0$;
> **Find:** a set of twins $\mathcal{T}$ such that each order belongs to exactly one twin ij of $\mathcal{T}$, and a starting time s_{ij} for each of these twins so that the number of on-time orders is maximized.

Note that the solution of $MDSP_A$ includes the possibility to schedule single-order trips (through twins jj, for $j \in J$).

3.1 Complexity

In this section we show that $MDSP_A$ is difficult. We make use of a reduction from the well-known (binary) $\mathcal{NP}$-complete decision problem:

Partition: Given q integers $\{a_1, a_2, \ldots, a_q\}$, is there a subset $S \subset \{1,\ldots,q\}$ such that $\sum_{i \in S} a_i = \frac{1}{2} \sum_{i=1}^{q} a_i$?

Theorem 2. *Problem $MDSP_A$ is $\mathcal{NP}$-hard.*

Proof. Consider an instance I of Partition with q integers $\{a_1, a_2, \ldots, a_q\}$ as above. Let $W = \sum_{i=1}^{q} a_i$. We can build a corresponding instance I' of $MDSP_A$ as follows.

In I' there are $n = 2(q+2)$ orders forming set $J = N \cup N'$ in which N and N' are two disjoint sets each with $(q+2)$ orders. In particular, for each integer i of the PARTITION instance I, we define a pair of identical *partner orders* $i \in N$ and $i' \in N'$ with $p_i = p_{i'} = a_i$, $r_i = r_{i'} = W - a_i$ and $d_i = d_{i'} = 2W + 1$, $i, i' = 1, \dots, q$. We refer to these orders as *item-orders*.

Moreover, there are two more pairs of partner orders. Namely, there is a pair of partner *short* orders $(q+1) \in N$ and $(q+1)' \in N'$, and a pair of partner *long* orders $(q+2) \in N$ and $(q+2)' \in N'$. All the data are reported in Table 1.

Table 1: Data of instance I'

order	Processing Time	Release Date	Due Date
$i, i' = 1, \dots, q$	$p_i = p_{i'} = a_i$	$r_i = r_{i'} = W - a_i$	$d_i = d_{i'} = 2W + 1$
$(q+1)$	$p_{(q+1)} = 1$	$r_{(q+1)} = \frac{W}{2} - 1$	$d_{(q+1)} = \frac{3}{2}W + 1$
$(q+1)'$	$p_{(q+1)'} = 1$	$r_{(q+1)'} = \frac{3}{2}W$	$d_{(q+1)'} = \frac{5}{2}W + 2$
$(q+2)$	$p_{(q+2)} = W$	$r_{(q+2)} = 0$	$d_{(q+2)} = 2W + 1$
$(q+2)'$	$p_{(q+2)'} = W$	$r_{(q+2)'} = W$	$d_{(q+2)'} = 3W + 1$

Clearly, in I' we have $\delta = W + 1$. In addition, the travel time between two partner orders i, i' is set to zero, $i, i' = 1, \dots, q+2$. For any other order-pair, the travel time is set to the maximum possible value that meets the triangle inequality. Hence, we have the following values for the travel times

$$t_{ji} = t_{ij} = \begin{cases} 0 & \forall i \in N,\ j = i' \in N'; \\ \frac{1}{2}(p_i + p_j) & \text{otherwise.} \end{cases}$$

which imply that the (feasible) twin processing times in I' are, from (3) and (5),

$$p_{ji} = p_{ij} = \begin{cases} p_i\ (= p_j) & \forall i \in N,\ j = i' \in N'; \\ p_i + p_j & \text{otherwise.} \end{cases}$$

It is easy to verify that any pair of orders $(i, j) \in J \times J$ forms a feasible twin (unless $i = (q+2)'$ and j is an item-order with $p_j > 1$.)

We next show that there is a schedule σ of $MDSP_A$ in which no order is tardy if and only if the instance of PARTITION is a yes-instance.

Let us consider a solution of $MDSP_A$ with no tardy orders. We next show that an early completion of *all* the orders implies that:

1. Orders $(q+2)$ and $(q+2)'$ do not form a twin.

2. Any order $i \in N \setminus (q+2)$ necessarily forms a twin with its partner order $i' \in N' \setminus (q+2)'$ and, symmetrically, any order $i' \in N' \setminus (q+2)'$ necessarily forms a twin with its partner order $i \in N \setminus (q+2)$.

To prove the first statement, consider first a schedule σ in which orders $(q+2)$ and $(q+2)'$ are in a twin. Such a twin must be processed between times W and $2W+1$. Consequently, all the $2q$ item-orders must complete not later than $W+1$ but, recalling the release and due dates of the item-orders, this is impossible for $q>2$. Hence, $(q+2)$ and $(q+2)'$ must form a twin.

To prove the second statement, suppose that a twin ij exists in σ in which i and j are not partner orders and i is not a long order. In this case, $p_{ij} = p_i + p_j$. In the total processing time $p(\sigma)$ of the orders in σ, p_i and p_j appear twice. In fact, the partner orders of i and j respectively may form a twin with some other order or be processed as singletons. In both cases, their duration is added to $p(\sigma)$. Then one would have:

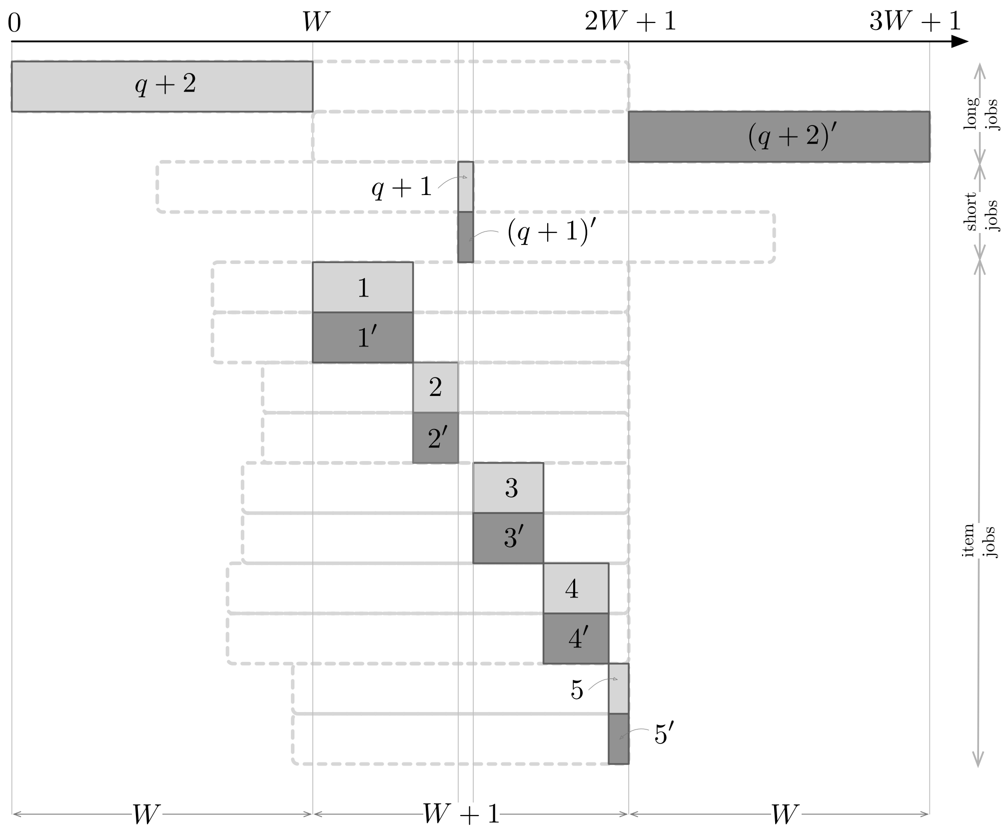
$$p(\sigma) \geq 2p_{(q+2)} + p_{(q+1)} + \sum_{\ell=1,\ldots,q} p_\ell + p_i + p_j = 2W + 1 + W + p_i + p_j > 3W + 1$$

which implies that there must be at least one order which is delivered after its due date, contradicting the hypothesis that no order is tardy in σ.

As a consequence of the previous facts, in any schedule with no tardy orders, the long orders $(q+2)$ and $(q+2)'$ are processed at the beginning and at the end of the schedule respectively, while the short orders $(q+1)$ and $(q+1)'$ form a twin that is necessarily processed in the interval $[\frac{3}{2}W, \frac{3}{2}W+1]$. The remaining item-orders form q twins with their respective partner order, and the processing times of each such twin equals the size of a PARTITION item. Moreover, those twins have only two disjoint intervals left for processing: $\mathcal{I}_1 = [W, \frac{3}{2}W]$ and $\mathcal{I}_2 = [\frac{3}{2}W+1, 2W+1]$. The resulting structure is illustrated in Figure 1.

Since the width of $\mathcal{I}_1$ and $\mathcal{I}_2$ is $\frac{W}{2}$, it is clear that a schedule with no tardy orders exists for instance I' if and only if I is a YES-instance of PARTITION. □

Figure 1: Illustration of a schedule corresponding to a YES-instance of PARTITION with five items. Dotted rectangles indicate the orders windows $[r_i, d_i]$.



4 Lower bounds

In this section we illustrate the lower bounds that we employed in our branch and bound algorithm. As observed in Cosmi et al. (2019a), the combinatorial lower bounds (presented in that work) outperform those provided by the Gurobi solver in 91% of the cases, and they are computed much faster. Below we present a combinatorial lower bound which improves those contained in Cosmi et al. (2019a).

Given an instance of $MDSP_A$, the idea is to define an auxiliary single-machine scheduling problem P_{aux} such that its optimal solution can be found efficiently and its value is a lower bound on the optimal value of $MDSP_A$. There is a one-to-one correspondence between orders in $MDSP_A$ and jobs in P_{aux}, therefore we use j to denote the job in P_{aux} corresponding to order j in $MDSP_A$. In particular, P_{aux} is an instance of the scheduling problem $1|r_j|\sum U_j$ in which release and due dates are *agreeable*, i.e., for any two jobs i and j such that $r_i < r_j$, then $d_i \leq d_j$. (For simplicity, we use the term *agreeable* either referred to a set of jobs or to the corresponding intervals.) This problem can be solved in $O(n^2)$ using the well known Kise-Ibaraki-Mine algorithm (KIM, Kise et al. (1978)), which generalizes the classical Moore's algorithm for $1||\sum U_j$ to the problem with agreeable release and due dates.

In the following, we describe how the auxiliary instance P_{aux} is defined. We first illustrate the definition of release dates and due dates, and then the definition of the processing times.

Definition of time windows. In what follows, we let $I_j = [r_j, d_j]$ denote the interval (time window) of order j in problem $MDSP_A$. We say that I_j is *properly contained* in I_i if $r_i < r_j$ and $d_i > d_j$. Note that a set of intervals is agreeable if and only if there are no two intervals I_i, I_j such that one is properly contained in the other.

In the instance of the auxiliary scheduling problem P_{aux}, for each order j in $MDSP_A$, we want to define a new interval $I'_j = [r'_j, d'_j]$ such that $I_j \subseteq I'_j$ and the intervals I'_j, $j = 1, \ldots, n$, are agreeable.

The idea is to obtain the intervals I'_j by enlarging the original intervals $[r_j, d_j]$. This can be done in many different ways. In particular, let us call *enlargement* the amount by which each interval $I_j = [r_j, d_j]$ is stretched. In order to preserve as far as possible the structure of the original intervals, it seems reasonable to determine the *minimum total enlargement* yielding an agreeable set of intervals. We next show how this can be attained.

For each job $j \in J$, let

$$\tilde{I}_j = \bigcup_{i: I_i \supset I_j} I_i \tag{13}$$

i.e., $\tilde{I}_j = [\tilde{r}_j, \tilde{d}_j]$ is the union of all the intervals that strictly contain I_j. Algorithm 1 computes the values of r'_j and d'_j. The idea is that each interval I_j is modified by either extending it leftwards (up to $\tilde{r}_j$) or rightwards (up to $\tilde{d}_j$).

Algorithm 1 Minimum Total Enlargement (MTE) Algorithm

for $j \in J$ **do**
  $\tilde{I}_j = [\tilde{r}_j, \tilde{d}_j] = \bigcup_{i: I_i \supset I_j} I_i$
  if $r_j - \tilde{r}_j < \tilde{d}_j - d_j$ **then**
    $r'_j := \tilde{r}_j$
  else
    $d'_j := \tilde{d}_j$
  end if
end for

Theorem 3. *The intervals returned by Algorithm 1 are agreeable and total enlargement is minimum.*

Proof. We first show that the intervals resulting from the application of the algorithm are agreeable. Consider any i, and suppose that we enlarge the interval I_i by setting the left endpoint of the interval to $\tilde{r}_i$. Now, the interval $I'_i = [\tilde{r}_i, d_i]$ is no more properly contained in any other interval.

We next observe that, as a consequence of the enlargement of I_i, even if I'_i is used instead of I_i in (13), no interval $\tilde{I}_j$ is affected, for any $j \neq i$. In fact, for this to happen, one should have that $I_j \subset I'_i$ and $\tilde{r}_i < \tilde{r}_j$, since otherwise no change in $\tilde{I}_j$ occurs. But this is not possible: In fact, since $I_j \subset I'_i$, then one would have $\tilde{I}_j \supseteq I'_i$ which in turn implies $\tilde{r}_i \geq \tilde{r}_j$.

Similar considerations hold if we enlarge the interval I_i by setting the right endpoint of the interval to $\tilde{d}_i$.

Finally, observe that total extension is minimum, since each interval I_i is extended by the minimum amount such that the extended interval I'_i is no more properly contained in any other interval. □

Note that a straightforward computation of the intervals $\tilde{I}_j$ requires time $O(n^2)$.

A more careful implementation of the above procedure is sketched hereafter. For every job j, consider the set

$$Q_j = \{i \in J \ : \ d_i > d_j \wedge r_i < r_j\}.$$

Clearly, if $Q_j = \varnothing$, there is no need to consider enlargements for j, otherwise $\tilde{r}_j = \min_{i \in Q_j}\{r_i\}$ and $\tilde{d}_j = \max_{i \in Q_j}\{d_i\}$. In order to determine such values, we sort the jobs in non-decreasing order of release dates and compute the set $\tilde{J}$ of jobs $h \in J$ with $Q_h = \varnothing$ (no other job $\ell \in J$

exists with $r_\ell < r_h$ and $d_\ell > d_h$). $\tilde{J}$ can be found in $O(n \log n)$ time by simply sweeping through the ordered set of jobs. Note that $\tilde{J} = \{h(1), h(2), \ldots, h(q)\}$ with $r_{h(\ell)} \le r_{h(\ell+1)}$ and $d_{h(\ell)} \le d_{h(\ell+1)}$, $\ell = 1, \ldots, q-1$.

For all $j = 1, \ldots, n$ we compute the minimum enlargements $\tilde{d}_j$ and $\tilde{r}_j$ as follows:

Let $\tilde{d}_0 = -\infty$. If $d_j > \tilde{d}_{j-1}$ then $\tilde{d}_j := d_j$ else $\tilde{d}_j := \tilde{d}_{j-1}$. As for the leftward enlargement $\tilde{r}_j$, we have to determine the job in Q_j with minimum release date. Then $\tilde{r}_j = \min\{r_i \ : \ d_i \ge d_j, i \in \tilde{J}\}$. That is, we search for the job $h \in \tilde{J}$ with minimum due date larger than d_j, which can be done in $O(\log n)$ time. Then $\tilde{r}_j = r_h$ (as job h has the minimum release date among those with due date greater than d_j.) The overall computational cost is $O(n \log n)$ and this is the complexity of the algorithm.

We next show how to define the processing times of the auxiliary instance P_{aux} of $1|r_j|\sum U_j$ with the new set of agreeable intervals I'_j, so that the value of the optimal solution is a lower bound on the optimal value of $MDSP_A$.

Definition of processing times. In order to have a lower bound on the original instance of $MDSP_A$, the processing time p'_j in P_{aux} must be defined in such a way that its value is not larger than the contribution of order j to the makespan of on-time orders in any solution of the instance of $MDSP_A$, regardless of whether order j forms a twin with some other order, or it is delivered in a single trip.

We consider two different ways of defining the processing times p'_j of the auxiliary instance P_{aux}.

1. *Flat lower bound.* We set the length of job j equal to

$$p'_j = t_j + \min\left\{t_j, \min_k \left\{\frac{1}{2} t_{jk}\right\}\right\}. \tag{14}$$

Note that $p'_j \le p_j$. Moreover, if we consider any twin ij in a feasible schedule of problem $MDSP_A$, one has $p'_i + p'_j \le t_i + t_j + t_{ij} = p_{ij}$. Then, given any solution σ of $MDSP_A$, with z tardy orders, if we sequence the jobs in P_{aux} in the same order in which the corresponding customers are visited in σ, we get a schedule with $z' \le z$ tardy jobs.

2. *Order-based lower bound.* Given the initial set of intervals in $MDSP_A$, we first apply Algorithm MTE. Then, we renumber all jobs in P_{aux} so that $i < j$ if $d'_i < d'_j$ or $d'_i = d'_j$ and $r'_i < r'_j$. The processing times p'_j are defined as follows:

$$p'_j = \begin{cases} p_j & j = 1 \\ \min\{p_j, \min_{\substack{h<j: \\ (h,j)\in D}} \{p_{hj} - p'_h\}, \min_{\substack{h<j: \\ (j,h)\in D}} \{p_{jh} - p'_h\}\} & j > 1 \end{cases} \tag{15}$$

Given an order j, we use the notation $m(j)$ to indicate the order such that:

$$p_{j,m(j)} = \min\{\min\{p_{ij} : ij \in D\}, \min\{p_{ji} : ji \in D\}, i \neq j\}.$$

If order j is not in any feasible twin, then $m(j)$ is undefined and $p'_j = p_j$.

Lemma 1. *In problem P_{aux}, for each job j it holds that (1) $p'_j \leq p_j$ and (2) $p'_j + p'_{m(j)} \leq p_{j,m(j)}$.*

Proof. Inequality (1) $p'_j \leq p_j$ follows from (15).

We next show that (2) holds. For each job j, either $m(j) = h < j$ or $m(j) = l > j$. We consider these two cases separately.

Case 1 $m(j) = h < j$.

It holds that: $p'_j = \min\{p_j, \min_{\substack{i<j: \\ (ij)\in D}} \{p_{ij} - p'_i\}\} \leq p_{hj} - p'_h$. Hence $p'_j + p'_h \leq p_{hj}$.

Case 2 $m(j) = l > j$

2.1 Consider the case in which p'_l is attained in correspondence to j, i.e., $j = \operatorname{argmin}_{\substack{i<l, \\ il\in D}} \{p_l, p_{il} - p'_i\}$. Hence, $p'_l \leq p_{jl} - p'_j$, so again $p_{jl} \geq p'_l + p'_j$.

2.2 If p'_l is attained for $k \neq j$, then: $j \neq k = \operatorname{argmin}_{\substack{i<l, \\ il\in D}} \{p_l, p_{il} - p'_i\}$. Hence, $p'_l = p_{kl} - p'_k \leq p_{jl} - p'_j$, which yields again $p'_l + p'_j \leq p_{jl}$.

This completes the proof. □

Theorem 4. *Each feasible schedule σ of value z_σ for the original problem $MDSP_A$ defines a sequence of jobs which corresponds to a feasible schedule σ' of value z'_σ for problem P_{aux} with $z_\sigma \geq z'_\sigma$.*

Proof. Given a feasible schedule σ for $MDSP_A$, we consider a schedule σ' for P_{aux} where:

- jobs are sequenced as the customers in σ;
- the processing time of job $j \in J$ is given by p'_j defined in (15);
- if job j corresponds to either a single order or to the first order in a twin in σ, then we let $s'_j = s_j$; if j is the second order of the twin ij in σ, then $s'_j = \max\{s'_i + p'_i, r'_j\}$.

Clearly, σ' is always a feasible schedule for P_{aux}, since $s'_j \geq r'_j$.

Given a feasible schedule σ for $MDSP_A$, we next show that the corresponding schedule σ' for P_{aux} has a number of tardy jobs which does not exceed the number of tardy orders in

σ. To this aim, we separately consider the two cases of single-order trips and twins. Recall that the values d'_j are obtained using the MTE Algorithm, hence $d'_j \geq d_j$ for each job $j \in J$.

Single-order trips Each job j corresponding to a single-order trip in which the order j is on time in σ, in σ' ends at $C'_j = s_j + p'_j \leq s_j + p_j \leq d_j \leq d'_j$, hence j is on time also in σ'.

Twins Let us consider a twin ij which is on time in σ, corresponding to the two consecutively scheduled jobs i and j in σ'. We consider separately the two jobs:

First Job The first job i starts at $s'_i = s_i$ and it ends at $C'_i = s_i + p'_i \leq s_i + p_i \leq d_i \leq d'_i$, hence i is on time in σ'.

Second Job In σ', job j starts either at its release time $r'_j \leq r_j$ or when the job i ends at C'_i. In the first case, since obviously $r_j + p_j \leq d_j$, and since $r'_j \leq r_j$, $p'_j \leq p_j$ and $d'_j \geq d_j$, one has $r'_j + p'_j \leq r_j + p_j \leq d_j \leq d'_j$. In the latter case, $C'_j = s_i + p'_i + p'_j$, and since, from Lemma 1, $p'_i \leq p_{i,m(i)} - p'_{m(i)} \leq p_{ij} - p'_j$, it follows that $p'_i + p'_j \leq p_{ij}$ and hence $C'_j \leq s_i + p_{ij}$, since the twin is on time in σ one has $s_i + p_{ij} \leq d_j$, so in conclusion $C'_j \leq d'_j$, i.e., also in this case job j is on time in σ'.

This completes the proof. □

Example 1. *We refer to an instance with* $n = 5$ *orders, namely* $1, 2, 3, 4, 5$. *Beside those corresponding to single orders, the feasible twins are* "12", "14", "21", "35", "41", "53". *Their processing times, release times, and due dates—obtained from Equations* (5), (6), *and* (7)*—are as follows:*

$$\{p_{ij}\} = \begin{pmatrix} 10 & 15 & - & 30 & - \\ 15 & 10 & - & - & - \\ - & - & 10 & - & 25 \\ 30 & - & - & 16 & - \\ - & - & 25 & - & 14 \end{pmatrix} \quad \{r_{ij}\} = \begin{pmatrix} 0 & 0 & - & 0 & - \\ 1 & 1 & - & - & - \\ - & - & 1 & - & 1 \\ 0 & - & - & 0 & - \\ - & - & 3 & - & 3 \end{pmatrix} \quad \{d_{ij}\} = \begin{pmatrix} 40 & 41 & - & 50 & - \\ 40 & 41 & - & - & - \\ - & - & 47 & - & 47 \\ 40 & - & - & 50 & - \\ - & - & 47 & - & 47 \end{pmatrix}$$

According to MTE Algorithm 1 and Equation (15) *we compute release times* (r'_j)*, due dates* (d'_j) *and processing times* (p'_j) *of the jobs in the instance of* P_{aux}*:*

Job j	1	2	3	4	5
r'_j	0	0	0	0	3
d'_j	40	41	47	50	50
p'_j	10	5	16	15	9

We solve P_{aux} using Kise-Ibaraki-Mine algorithm obtaining the following schedule: $\sigma' = \langle 1, 2, 4, 5 \rangle$ *which provides a lower bound equal to 1 (job 3 is late). Note that, there is a feasible (and optimal) schedule for the original instance of the problem* $\sigma = \langle \text{"12"}, \text{"35"} \rangle$ *in which job 4 is late.*

5 A combinatorial branch and bound

Here we describe the combinatorial branch and bound (B&B) we have developed to solve the problem $MDSP_A$.

B&B works as follows. As in many enumeration algorithms for scheduling problems, we fix the orders from left to right, taking into account that there are two types of trips, namely single-order trips and twins. Each node l of the enumeration tree is characterized by:

- a set S_l of on-time scheduled orders;
- a schedule σ_l of the orders in S_l (called *partial schedule*), specifying single-order trips and twins;
- a set T_l of tardy orders;
- the set of unscheduled orders $J_l = J \setminus (S_l \cup T_l)$.

At each node l of the enumeration tree, children are generated by appending an order to the partial schedule σ_l, forming either a single-order trip or a twin with the currently last order. More in detail, let k be the last scheduled order in σ_l. Consider two cases.

(i) k is scheduled in a single-order trip. In this case, each child node is obtained by appending to σ_l an order h scheduled either as a single-order trip or as the second order of the twin kh.

(ii) k is the second order of a twin ik. In this case, each child node is obtained by appending an order scheduled as a single-order trip.

For example, in Figure 2, node 5 of the enumeration tree is a child of node 2 and corresponds to a partial schedule consisting of two single-order trips, namely $\sigma_5 = \langle b, c \rangle$. The sibling node 6 corresponds instead to a partial schedule with one twin $\sigma_6 = \langle bc \rangle$.

Consider a partial schedule σ_l, and an order $h \in J_l$ such that, when appended to σ_l, h is tardy. Then, h will be inserted in the set T_q for all nodes q of the subtree rooted in l.

When branching from node l, its children nodes are built taking into account some dominance rules:

- In case (i) above, for each $h \in J_l$, a child node is generated appending to σ_l the order h only if the following conditions are met:
  - $\nexists j \in J_l : \max(r_j, s_k + p_k) + p_j \leq r_h$
  - $\nexists j \in J_l : \max(r_{kj}, s_k) + p_{kj} \leq r_h$
  - if h is scheduled as a single-order trip, $s_k + p_k + p_h \leq d_h$
  - if h is scheduled as the second order of the twin $kh \in D$, $\max(s_k, r_{kh}) + p_{kh} \leq d_{kh}$.
- In case (ii) above, i.e., l ends with the twin ik having starting time s_{ik}, for each order $h \in J_l$, a child node is generated appending to σ_l the order h only if the following condition is met:
  - $\nexists j \in J_l : \max(s_{ik} + p_{ik}, r_j) + p_j \leq r_h$.

Lower Bounds.

At each node l of the enumeration tree, a lower bound is computed applying Algorithm 2. If the computed lower bound is not smaller than the incumbent best solution, the node is fathomed.

Algorithm 2 Twin Lower Bound

BUILD PROBLEM P:
Use MTE Algorithm on the job set J to compute release times and due dates
Build EDD sequence of jobs in J and rename jobs according to it
for $j \in \{1, \dots, |J|\}$ **do**
    if $j = 1$ **then**
        $\tilde{p}_j = p_j$
    else $j \neq 1$
        $\tilde{p}_j = \min\{p_j, \min_{\substack{h<i:\\(h,j)\in D}} \{(p_{hj} - \tilde{p}_h\}, \min_{\substack{h<j:\\(j,h)\in D}} \{(p_{jh} - \tilde{p}_h\}\}$
    end if
end for
SOLVE P:
Run KIM algorithm (Kise et al. (1978))

The procedure to compute the lower bound is slightly different in cases (i) and (ii) above. In case (ii), the partial schedule σ_l ends with a twin ik. Let t^* be the completion time of the twin ik. In this case, Algorithm 2 is applied to the set J_l of unscheduled orders, considering that no trip can start before t^*.

On the other hand, if case (i) holds, i.e., σ_l ends with the single-order trip k, then Algorithm 2 is applied to the set $J_l \cup \{k\}$, and the completion time of the order preceding k in σ_l is the first available starting time for the other trips. This is due to the fact that k can still be the first order in a twin ki with $i \in J_l$, so it has to be reconsidered when computing the lower bound.

In both cases, if we let z_l be the value obtained applying Algorithm 2, the lower bound at node l is $|T_l| + z_l$.

Upper bounds. Before children are generated from a node l, in order to speed up the algorithm, a heuristic is run to compute a suitable upper bound and possibly update the incumbent solution.

In the heuristic, single-order trips and twins are considered. The idea is to sequence the unscheduled orders (J_l) by iteratively computing a weight for each order or twin and selecting as the next order or twin the one having minimum weight.

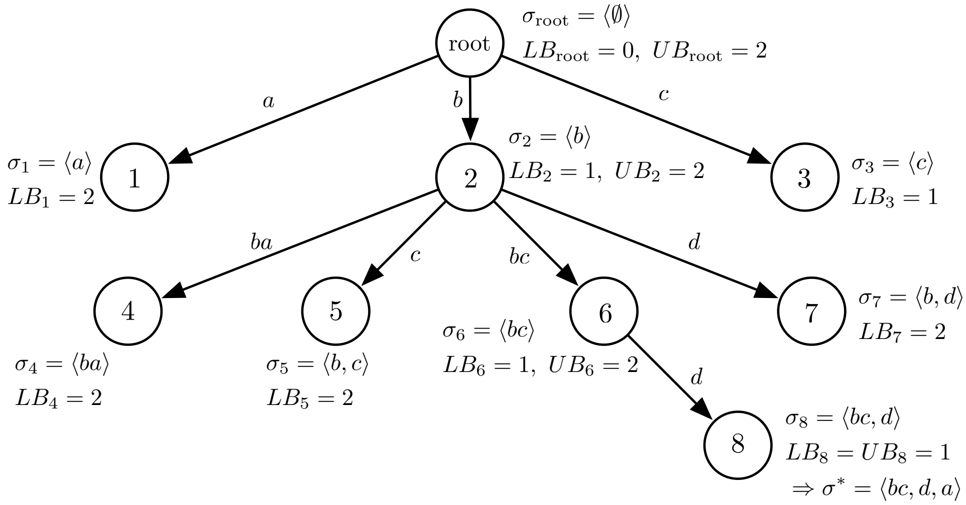
Let t^* be the current makespan, i.e., the completion time of the last order k in σ_l. Let C_j denote the completion time of job $j \in J_l$ if it is scheduled immediately after k in a single-order trip, and $C_j \leq d_j$. In this case, we let $w_j = C_j - t^*$. If j is appended to k to form the twin $kj \in D$, completing at $C_{kj} \leq d_{kj}$, we let $w_{kj} = C_{kj} - t^*$. Finally, if a twin ij is appended to k, completing at $C_{ij} \leq d_{ij}$, we let $w_{ij} = \frac{C_{ij} - t^*}{2}$. In all cases, if the completion time of the single-order trip or twin exceeds the due date, we assume that the weight is $+\infty$. Once a new single-order trip or twin is added to the partial schedule, it is considered as the new last trip, t^* is recomputed and the algorithm iterates until the last available order is scheduled. Note that orders/twins having a large release time may receive a large weight even if they are short and their due date is smaller compared to other longer orders.

The branch and bound tree is explored using a depth-first strategy. Among the sibling nodes of a given node, the node having the lowest lower bound is selected. In case of a tie, the node l is selected in which the difference between due date and completion time of the last order of σ_l is smallest.

For the sake of clarity, the following example gives an idea of our *ad-hoc* branching procedure.

Example 2. *We refer to an instance with $n = 4$ orders, namely a, b, c, d. Feasible twins are ab, ac, ba, bc, cb (while twins ca and jd, dj with $j = a, b, c$, are not feasible). The branch and* *bound tree is shown in Figure 2. Beside each node l, we report the partial schedule σ_l and the value computed for the lower LB_l and, possibly upper UB_l bounds. At the root node, a lower*

Figure 2: An example of the enumeration tree of the branch and bound procedure.



bound $LB_{\text{root}} = 1$ and an upper bound $UB_{\text{root}} = 2$ (which is therefore the incumbent solution schedule) are computed. Children 1, 2, and 3 of the root node are generated and correspond to scheduling orders a, b, and c, respectively, at the beginning of the schedule. The child node relative to the partial schedule $\langle d\rangle$ is not generated due to the dominance rule (order b can be delivered before d is released). Concurrently, lower bounds at each node are computed.

A depth-first strategy is applied and node 2 is explored first since it has the lowest lower bound (together with node 3). An upper bound UB_2 = Nodes 4, 5, 6, and 7 are then generated: node 6 shows a lowest lower bound $LB_6 = 1$. Now, node 6 is explored and its child node 8 is generated and, in turn, explored: Here, the heuristic computes a new incumbent solution with value $UB_8 = 1$ and hence, nodes 1, 3, 4, 5, and 7 are fathomed. The corresponding schedule $\sigma^* = \langle bc, d, a\rangle$ (in which order a is late) is an optimal schedule.

6 Computational results

In this section we present the results of a large computational campaign including three distinct experimental settings. [Instances are available from the authors upon request.]

1. (*Real-life offline scenario.*) The first experimental setting consists of real data from a case study, in which all orders have been already accepted, and deliveries must be

scheduled. As orders have been accepted by some rough-cut capacity planner, this scenario typically results in a relatively small number of tardy deliveries.

2. (*High-demand offline scenario.*) The second setting consists in randomly generated instances in which the distribution of ideal delivery times is close to that of real-life instances, but the number of orders has been increased. Also in this scenario, orders have already been accepted. The purpose of this experiment is to test our solution algorithms in a more stressed scenario than the previous scenario.

3. (*Real-life online scenario.*) This setting is based on the real data of scenario 1, and it is the closest to real-life operations. Customer orders are supposed to arrive over time, and the overall schedule is updated at regular intervals in a rolling horizon fashion.

We have addressed all the instances of both real-life and random scenarios using the combinatorial branch and bound (B&B) algorithm presented in Section 5 and an ILP formulation for the problem. In particular, we used the ILP formulation which turned out to be the most effective among a set of possible formulations in Cosmi et al. (2019b). The formulation is reported in the Appendix.

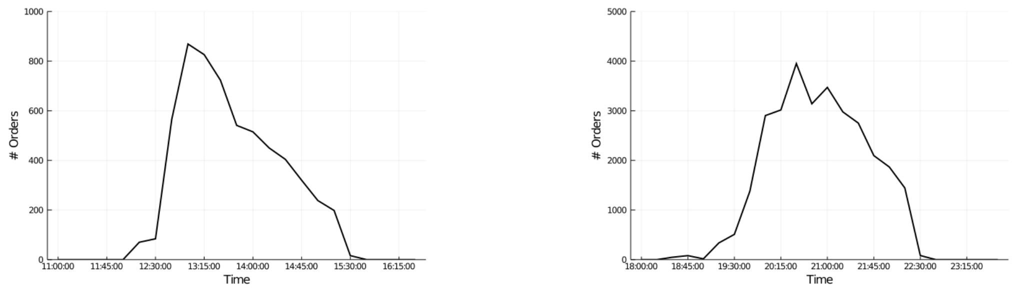
Our experiments were aimed at investigating (i) the benefits of order aggregation with respect to having single-order deliveries, (ii) the computational efficiency of the two solution methods and (iii) the effectiveness of our approach in practice. In particular, we investigate (i) in Scenario 1, (ii) in Scenarios 1 and 2, and (iii) in Scenario 3.

All tests were performed on a computer equipped with an Intel Xeon E5-2643v3 3.40 GhZ CPU and 32 GB RAM. B&B was implemented using Julia (Bezanson et al. (2017).) The ILP model is built using JuMP (Dunning et al. (2017)) and it is solved using Gurobi 9.0 (Gurobi Optimization LLC (2018)) in its multithread version. The time limit is set to one hour, for both B&B and Gurobi.

6.1 Real-life offline scenario

The first experimental scenario consists of 478 instances derived from data provided by an Italian food delivery operator. In each instance, the number of orders n varies from 5 to 31 orders. The time horizon is roughly a courier shift, i.e., 4 hours, and corresponds, in all instances, to the busiest hours of the day, namely the hours spanning either lunch time or dinner time. Instances refer to nine different restaurants and a number of different days, therefore they may vary in physical location of the customers and distribution of ideal

Figure 3: Distribution of ideal delivery times in real-life instances.



(a) Lunch

(b) Dinner

delivery times. The distributions of ideal delivery times around lunch time and dinner time are shown in Figure 3 (a) and (b) respectively.

In Table 2, for each n between 5 and 31, the number of instances with n orders is reported.

The delivery time window of each order i is centered around the ideal delivery time $\hat{d}_i$, and the slack is $\delta = 30$ minutes. This corresponds to assuming that a delivery is acceptable as long as its earliness or tardiness with respect to $\hat{d}_i$ does not exceed 15 minutes, which corresponds to a commonly accepted level of service in food delivery.

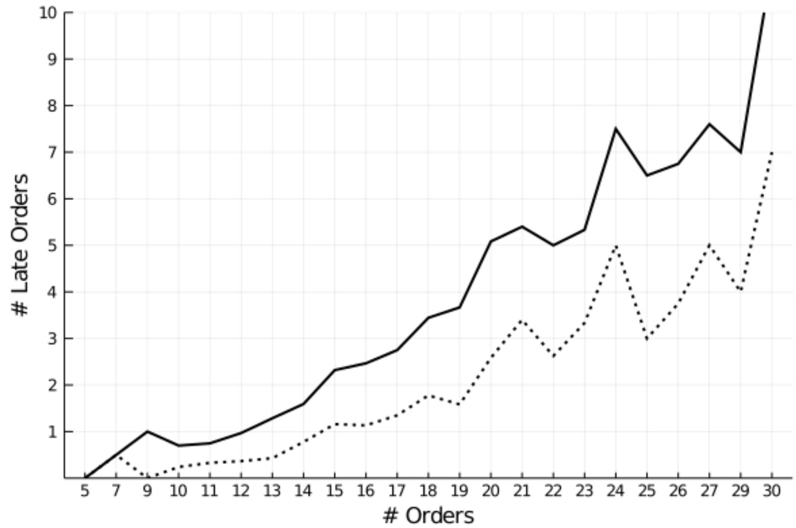
While the analysis of the effectiveness of our solution approach is presented in Section 6.1.2, we next comment on the values of the number of late order with and without order aggregation.

6.1.1 Benefits of order aggregation

Aggregating two orders in the same delivery represents an opportunity for increasing the productivity of the couriers with respect to single-order deliveries. It is therefore interesting to evaluate the improvement, in terms of late deliveries, that can be achieved by order aggregation. For each of the instances of the real-life scenario solved to optimality, we compared the number of late orders with and without the possibility of order aggregation. (The problem with single-order deliveries has been solved using an ILP formulation similar to the one in the Appendix) The results are summarized in Figure 4, in which, for each value of n, the average number of late deliveries is displayed with and without order aggregation.

From Figure 4 we observe that order aggregation consistently results in a lower number of late deliveries, typically saving 1 or 2 late deliveries for medium-size instances. A detailed comparison of the results show that in 212 out of 476 optimally solved instances, order aggregation reduces the number of late deliveries (compared to the single-order setting).

Figure 4: Average number of late deliveries in real-life instances when order aggregation is allowed (dashed line) and when it is not (solid line).



If we compare the total number of late deliveries (across all the 476 instances) with and without order aggregation, it turns out that with order aggregation, such a total number of late deliveries is 47.91% smaller.

6.1.2 Comparison between the efficiency of the solution approaches

In this section we analyze the computational behavior of the branch and bound approach in the real-life scenario, and compare it to the ILP formulation in Appendix.

The results are reported in Table 2. The 478 instances are grouped for different values of n. For each row, columns 3–6 report figures over all the instances having a certain value of n, namely: the number of instances for which B&B succeeded to find the optimal solution within the time limit (column 3), the number of instances for which the ILP solver found the optimal solution within the time limit (column 4), the average CPU time of B&B (column 5) and of the ILP solver (column 6) on the instances that were optimally solved within the time limit by the respective algorithm. The second last row reports the total number of unsolved instances, and the average CPU times on solved instances with the two methods respectively. The last row only considers the 445 instances that have been solved to optimality by both methods.

B&B solves to optimality all the instances with up to 30 orders within 1 hour of CPU, while ILP leaves 33 instances unsolved. B&B performs better also in terms of average running time. If we consider only the 445 instances optimally solved by both methods, the average running time for B&B is 5.99 secs, hence on the average B&B is 8 times faster than Gurobi on these instances.

Table 2: Results of B&B vs ILP for real-life instances. *CPU time refers to solved instances only.

n	# Inst	B&B solved	ILP solved	B&B CPU sec.	ILP CPU sec.
5	1	1	1	0.60	0.01
7	2	2	2	0.61	0.03
9	2	2	2	0.00	0.07
10	113	113	113	0.14	0.14
11	91	91	91	0.36	0.19
12	63	63	63	0.37	0.48
13	42	42	42	0.23	0.76
14	27	27	27	3.87	8.80
15	25	25	25	2.49	11.18
16	15	15	14	1.36	3.41*
17	20	20	20	23.90	124.32
18	18	18	16	39.06	82.13*
19	12	12	11	13.46	307.38*
20	12	12	8	92.76	326.89*
21	5	5	2	6.30	908.44*
22	8	8	5	8.07	490.32*
23	3	3	2	291.02	2743.12*
24	2	2	0	483.53	-
25	2	2	0	390.77	-
26	4	4	1	612.17	1590.40*
27	5	5	0	669.20	-
29	2	2	0	73.57	-
30	2	2	0	437.56	-
31	2	0	0	-	-
Overall	478	476	445	25.76	49.00
Solved by both		*445*		*5.99*	*49.00*

6.2 High-demand scenario

In this section we report the computational results concerning a set of instances which are generated by suitably modifying and scaling real-life instances. The purpose of this experiment is to evaluate the performance of the two solution approaches in a realistic, busier scenario characterized by a larger number of orders, which therefore results in larger problems. In particular:

- Restaurant locations are obtained by slightly perturbing their original positions (this has been done in order to conceal the identity of the restaurants.)
- The number n of orders is chosen in $\{15, 25, 35, 45\}$.
- Customer ideal delivery times are generated using two different distributions, namely Gaussian and Uniform, with parameters adapted from the real distributions in Fig. 3 (a) and (b). More precisely, the Gaussian distribution has the same mean and variance of the real distributions, while the Uniform distribution has the same mean value and spans from 11.30 to 15.30 for lunchtime and from 18.30 to 22.30 for dinnertime.

In conclusion, we have 8 different experimental settings, each setting being characterized by a value for n and delivery time distribution either Gaussian or Uniform.

For each setting, 90 instances have been generated. For each instance, the orders are generated in points which are uniformly scattered throughout a region defined as the smallest rectangle including all the orders from the restaurants. The location of the orders determines the processing times p_j and travel times t_{ij}. The results are shown in Tables 3–6.

A few comments are in order.

- On the whole, the combinatorial branch and bound algorithm has a superior performance with respect to the ILP. In particular, almost all the instances up to 25 orders are solved to optimality by B&B within the time limit, while the ILP solver runs into trouble for more than 15 orders and even for $n = 15$ in the Gaussian scenario. Comparing the CPU times on the instances for which both methods found the optimal solution, B&B turns out to be almost two orders of magnitude faster than ILP.
- In terms of CPU time, the performance of B&B is not significantly different on these instances with respect to real-life instances. On the contrary, even for the instances with $n = 15$ solved to optimality, the ILP solver requires significantly more time (322.95

seconds for Uniform instances, 876.1 seconds for Gaussian instances) than in the real-life instances (11.18 seconds). In other words, B&B appears more suitable to deal with congested instances than the ILP.

- Tables 5 and 6 allow a comparison between B&B and ILP also including the instances which were not solved to optimality. In these tables, Δ denotes the average percentage difference between the values of the best found solution by the two methods, UB_{ILP} and $UB_{B\&B}$ respectively. The tables distinguish the instances for which neither method was able to certify optimality (Unsolved instances) and the instances for which only B&B certified optimality. For instances unsolved by both methods, we report the number of instances for which, respectively, B&B found a better solution ($\Delta > 0$), the two best found solutions have the same value ($\Delta = 0$), and the ILP incumbent was better ($\Delta < 0$). For the instances for which B&B certified optimality, we report the number of instances for which $\Delta > 0$, the number of instances for which $\Delta = 0$ and finally for which $\Delta = 0$ and the ILP was able to certify optimality. On the whole, we observe that only in 49 (out of 720) instances did ILP find a better solution than B&B, while in 221 instances the solution found by B&B (either certified optimal or not) was strictly better than the incumbent of the ILP solver when the time limit was reached. We also notice that for $n = 15$ the ILP indeed always found the optimal solution, even if it was not always able to certify it, but as n grows the superiority of B&B is increasingly apparent, even in unsolved instances.

Table 3: Results of B&B for Gaussian instances. *CPU time refers to solved instances only. †The B&B gaps refer to unsolved instances only.

n	# Inst	B&B solved	ILP solved	B&B gap (%)†	B&B CPU sec.	ILP CPU sec.
15	90	90	40	-	2.85	876.10*
25	90	88	0	48.33	188.51*	-
35	90	50	0	31.15	1054.33*	-
45	90	1	0	31.27	979.59*	-
Overall	360	229	40	31.50	308.04*	876.10*
Solved by both		*40*			*21.92*	*876.10*

Table 4: Results of B&B for Uniform instances. *CPU time refers to solved instances only. †The B&B gaps refer to unsolved instances only.

n	# Inst	B&B solved	ILP solved	B&B gap (%)†	B&B CPU sec.	ILP CPU sec.
15	90	90	90	-	4.40	322.95
25	90	89	2	38.46	358.51*	1365.37*
35	90	14	0	32.07	1985.94*	-
45	90	0	0	31.93	-	-
Overall	360	193	92	32.03	311.43*	345.61*
Solved by both		*92*			*8.18*	*345.61*

Table 5: Comparison between best found solutions: Gaussian instances ($\Delta = UB_{ILP} - UB_{B\&B}$).

n	Unsolved Instances					Instances solved by B&B			
	$\Delta > 0$		$\Delta = 0$	$\Delta < 0$		$\Delta > 0$		$\Delta = 0$	$\Delta = 0$: OPT
	%	#	#	%	#	%	#	#	#
15	-	-	-	-	-	-	-	50	40
25	-	-	2	-	-	7.41	17	71	-
35	6.60	19	16	-4.28	5	5.39	29	21	-
45	4.63	42	42	-5.50	5	-	-	1	-
Overall	5.24	61	60	-4.89	10	6.14	46	143	40

Table 6: Comparison between best found solutions: Uniform instances ($\Delta = UB_{ILP} - UB_{B\&B}$).

n	Unsolved Instances					Instances solved by B&B			
	$\Delta > 0$		$\Delta = 0$	$\Delta < 0$		$\Delta > 0$		$\Delta = 0$	$\Delta = 0$: OPT
	%	#	#	%	#	%	#	#	#
15	-	-	-	-	-	-	-	-	90
25	-	-	-	-8.33	1	7.81	30	57	2
35	6.12	31	30	-6.74	15	6.14	11	3	-
45	4.90	42	25	-4.46	23	-	-	-	-
Overall	5.42	73	55	-5.43	39	7.49	41	60	92

6.3 Real-life online scenario

Here we report the computational results for the third scenario. Besides the order information in Scenario 1, here we also consider the time at which each order has been placed.

The scheduler operates by solving subsequent instances of the problem, at certain *decision times*. At each decision time, the set of orders considered includes all orders received and not yet delivered. After a solution is computed for the current set of orders, it is implemented up to the next decision time. If a decision time occurs at time t, the next decision time occurs the first time the courier returns to the restaurant after time $t + T$. The choice of T is an important parameter. If T is too small, at each decision point the set of released orders considered by the algorithm can be too small, possibly missing many optimization opportunities. On the other hand, if we wait too long to run the next instance of the problem (i.e., if T is too large), the orders' time windows may become too tight, and there may be no time to schedule an order which might have been accommodated earlier.

The average number of late orders (i.e., order that will have to be outsourced, as discussed in Section 1) throughout the whole set of real-life instances when solved offline is 0.85. In our experiments, we have run the online scenario employing the values $T = 5, 10, 15, 20, 25, 30$. The results of these experiments are reported in Table 7. Each row shows the average value of the number of late orders on the whole set of 478 real-life instances.

T	5	10	15	20	25	30
#	2.16	2.12	2.05	2.03	2.08	2.16

Table 7: Average number of late orders in the 478 real-life instances for various values of T.

The best value for T is $T = 20$, yielding an average of 2.03 late orders. Hence, the fact that orders are not all known at the beginning but rather arrive over time results in an average of 1.18 extra orders which have to be outsourced at each shift. Thanks to the limited number of jobs considered at each run, the solution algorithm at each decision time requires few seconds, hence making the approach viable in practice.

7 Conclusions

In this paper we addressed a single-restaurant, single-courier last-mile-delivery scheduling problem in which the objective is the maximization of on-time deliveries. To improve the quality of the solutions we consider the possibility of aggregating two orders, so that a single courier can deliver one or two meals in the same trip.

Our experiments show that order aggregation allows to improve the quality of service, significantly decreasing the number of late deliveries (on a sample of real-life data). Moreover, the relatively short computation times for small-sized instances suggest that the model can also be usefully employed in a dynamic setting, i.e., a set of new incoming orders may trigger the solution of a new instance in which the newly released orders are added to the orders not yet delivered.

The proposed branch and bound algorithm outperforms the best ILP model presented in Cosmi et al. (2019b) both in terms of effectiveness and efficiency: Within the time limit of one hour, B&B solves instances with up to 45 orders which is about twice the size of the largest instances solved by the ILP within the same time limit. Concerning the instanced solved by both methods, B&B is from 10 to 40 times faster than the ILP solver.

Possible directions of research may include (but are not limited to): (*i*) Investigating this last-mile delivery problem in a robust optimization setting in which uncertainty affects the data concerning courier travel times and/or waiting times at the restaurant. In this respect, our model would present similarities to flexible maintenance scheduling problems as, for instance, in Aloulou and Della Croce (2008), Detti et al. (2019). (*ii*) The design of a branch and price ad-hoc algorithm using either a time-indexed formulation or a packing formulation (see, e.g., Agnetis et al. (2009)). (*iii*) Addressing a last-mile delivery problem in which the same courier is shared by multiple restaurants, which brings the problem closer to a multi-agent scheduling problem where different agents (the restaurants) compete for the usage of a single machine (the courier, see e.g. Agnetis et al. (2013, 2015), Nicosia et al. (2018)). (*iv*) While our results have been devised under the no-wait assumption (see Section 3), the problem can be analyzed for different definitions of a feasible twin. For instance, the courier can be allowed some limited slack time between two deliveries in the same trip, or the twin is considered feasible if the second order is delivered within a certain time from the departure from the restaurant.

References

Agnetis, A., Alfieri, A., and Nicosia, G. (2009). Single-machine scheduling problems with generalized preemption. *INFORMS Journal on Computing*, 21(1):1–12.

Agnetis, A., Nicosia, G., Pacifici, A., and Pferschy, U. (2013). Two agents competing for a shared machine. In Perny, P., Pirlot, M., and Tsoukiàs, A., editors, *Algorithmic Decision Theory*, pages 1–14, Berlin, Heidelberg. Springer Berlin Heidelberg.

Agnetis, A., Nicosia, G., Pacifici, A., and Pferschy, U. (2015). Scheduling two agent task chains with a central selection mechanism. *Journal of Scheduling*, 18(3):243–261.

Aloulou, M. A. and Della Croce, F. (2008). Complexity of single machine scheduling problems under scenario-based uncertainty. *Operations Research Letters*, 36(3):338 – 342.

Archetti, C., Feillet, D., and Speranza, M. G. (2015). Complexity of routing problems with release dates. *European Journal of Operational Research*, 247(3):797 – 803.

Azi, N., Gendreau, M., and Potvin, J.-Y. (2012). A dynamic vehicle routing problem with multiple delivery routes. *Annals of Operations Research*, 199(1):103–112.

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98.

Böhm, M., Megow, N., and Schlöter, J. (2021). Throughput scheduling with equal additive laxity. In Calamoneri, T. and Corò, F., editors, *Algorithms and Complexity*, pages 130–143, Cham. Springer International Publishing.

Cieliebak, M., Erlebach, T., Hennecke, F., Weber, B., and Widmayer, P. (2004). Scheduling with release times and deadlines on a minimum number of machines. In Levy, J.-J., Mayr, E. W., and Mitchell, J. C., editors, *Exploring New Frontiers of Theoretical Informatics*, pages 209–222, Boston, MA. Springer US.

Cosmi, M., Nicosia, G., and Pacifici, A. (2019a). Lower bounds for a meal pickup-and-delivery scheduling problem. In *Proceedings of the 17th Cologne-Twente Workshop on Graphs and Combinatorial Optimization, CTW 2019*, pages 33–36.

Cosmi, M., Nicosia, G., and Pacifici, A. (2019b). Scheduling for last-mile meal-delivery processes. In *IFAC-PapersOnLine*, volume 52, pages 511–516.

Cosmi, M., Oriolo, G., Piccialli, V., and Ventura, P. (2019c). Single courier single restaurant meal delivery (without routing). *Operations Research Letters*, 47(6):537 – 541.

Detti, P., Nicosia, G., Pacifici, A., and Zabalo Manrique de Lara, G. (2019). Robust single machine scheduling with a flexible maintenance activity. *Computers & Operations Research*, 107:19 – 31.

Dunning, I., Huchette, J., and Lubin, M. (2017). Jump: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320.

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

Gurobi Optimization LLC (2018). Gurobi optimizer reference manual.

Kise, H., Ibaraki, T., and Mine, H. (1978). A solvable case of the one-machine scheduling problem with ready and due times. *Operations Research*, 26(1):121–126.

Klapp, M. A., Erera, A. L., and Toriello, A. (2018a). The dynamic dispatch waves problem for same-day delivery. *European Journal of Operational Research*, 271(2):519–534.

Klapp, M. A., Erera, A. L., and Toriello, A. (2018b). The one-dimensional dynamic dispatch waves problem. *Transportation Science*, 52(2):402–415.

Nicosia, G., Pacifici, A., and Pferschy, U. (2018). Competitive multi-agent scheduling with an iterative selection rule. *4OR*, 16(1):15–29.

Ulmer, M. W., Goodson, J. C., Mattfeld, D. C., and Thomas, B. W. (2020). On modeling stochastic dynamic vehicle routing problems. *EURO Journal on Transportation and Logistics*, 9(2):100008.

Ulmer, M. W., Thomas, B. W., and Mattfeld, D. C. (2019). Preemptive depot returns for dynamic same-day delivery. *EURO Journal on Transportation and Logistics*, 8(4):327–361.

van Bevern, R., Niedermeier, R., and Suchý, O. (2017). A parameterized complexity view on non-preemptively scheduling interval-constrained jobs: few machines, small looseness, and small slack. *Journal of Scheduling*, 20(3):255–265.

Acknowledgements

The authors wish to thank Ulrich Pferschy for suggesting an improvement in the complexity of Algorithm MTE.

Appendix: ILP model for $MDSP_A$

In this section we report the ILP model used to benchmark the branch and bound algorithm. This formulation has been presented in Cosmi et al. (2019b) along with three other integer programs for $MDSP_A$, and it was shown to be the most effective among the four ILPs.

In the objective function, we count the number of late orders by using indicator variables $y_{ij} \in \{0, 1\}$ representing whether a twin ij is late ($y_{ij} = 1$) or not ($y_{ij} = 0$). We account for the fact that a late twin order ij is assumed to be equivalent to both i and j late[1] by setting, for each $ij \in D$, a parameter $w_{ij} = 2$ if $i \neq j$, i.e., if ij is a twin order, and $w_{ij} = 1$ if $i = j$.

[1]This is with no loss of generality: In fact, suppose a schedule σ exists in which a twin ij has one order on time and the other one late. Then σ is dominated by another schedule $\bar{\sigma}$ identical to σ but for i and j split into two single orders. In this case only one between i and j is on time and the value of the objective function is such that $f(\bar{\sigma}) \leq f(\sigma)$.

We define the following decision variables:

- $s_i \in \mathbb{R}_+$, representing the starting time of order i;
- $x_{ij} \in \{0,1\}$, encoding whether order i precedes order j;
- $\beta_{ij} \in \{0,1\}$, indicating whether orders i and j are scheduled as a twin order ij.

The formulation is illustrated below.

$$
\begin{aligned}
\min\ & \sum_{ij \in D} w_{ij} y_{ij} && && (16)\\
s.t.\ & s_i \geq \sum_{ij \in D} r_{ij} \beta_{ij} && i \in J && (17)\\
& s_j \geq c_i - M(\beta_{ij} - x_{ij} + 1) && i,j \in J : i \neq j && (18)\\
& s_i \geq c_j - M(\beta_{ij} + x_{ij}) && i,j \in J : i \neq j && (19)\\
& s_j \geq s_i - M(1 - \beta_{ij}) && ij \in D && (20)\\
& s_i + p_{ij} \leq d_{ij} + M(y_{ij} + 1 - \beta_{i,j}) && ij \in D && (21)\\
& s_j + p_{ij} \leq d_{ij} + M(y_{ij} + 1 - \beta_{i,j}) && ij \in D && (22)\\
& c_j = s_j + \sum_{i:ij \in D} p_{ij} \beta_{ij} && j \in J && (23)\\
& x_{ij} + x_{ji} = 1 && ij : i \neq j \in D && (24)\\
& x_{jh} \geq x_{ih} - M(1 - \beta_{ij}) && \begin{array}{c} ij \in D \\ h \in J : h \neq i,j \end{array} && (25)\\
& \sum_{j:ij \in D} \beta_{ij} \leq 1 && i \in J && (26)\\
& \sum_{i:ij \in D} \beta_{ij} \leq 1 && j \in J && (27)\\
& \sum_{j:ij \in D} \beta_{ij} + \sum_{h:hi \in D, h \neq i} \beta_{hi} = 1 && i \in J && (28)\\
& s_i, c_i \in \mathbb{R}_+ && i \in J && (29)\\
& \beta_{ij}, x_{ij}, y_{ij} \in \{0,1\} && ij \in D && (30)
\end{aligned}
$$

Constraints (26)-(28) impose that each order i must always be associated and hence scheduled within a twin order (including order (i,i)). Constraints (17) force each order i to always start after the release time of the twin order it is associated to. Constraints (18)-(19) are standard disjunctive constraints imposing precedences between each pair of orders i and

j not belonging to the same twin order. These constraints do not hold if i is scheduled as single order, whereas, in the latter case, constraint (20) holds. Each single order has to be completed before its twin order delivery time otherwise the twin order is considered late (21)-(22). Constraints (24) impose that if i precedes j then it is not possible that j precedes i. Constraint (25) sets that if i and j are scheduled in the twin order ij then if i precedes h also j has to precede h.