Computers & Industrial Engineering 183 (2023) 109514

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Two is better than one? Order aggregation in a meal delivery scheduling problem☆

Alessandro Agnetis [a,*], Matteo Cosmi [b], Gaia Nicosia [c], Andrea Pacifici [d]

[a] Universita' degli Studi di Siena, Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche, via Roma 56, 53100, Siena, Italy
[b] University of Luxembourg, Luxembourg Centre for Logistics and Supply Chain Management, 6, rue Richard Coudenhove-Kalergi, 1359, Luxembourg, Luxembourg
[c] Universita' degli Studi Roma Tre, Dipartimento di Ingegneria Civile, Informatica e delle Tecnologie Aeronautiche, Via della Vasca Navale 79, 00146, Rome, Italy
[d] Universita' degli Studi di Roma Tor Vergata, Dipartimento di Ingegneria Civile e Ingegneria Informatica, via del Politecnico 1, 00133, Rome, Italy

ARTICLE INFO

Keywords:
Scheduling
Complexity
Branch and bound
Last-mile meal delivery

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 $\pm\frac{\delta}{2}$, 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 $\mathcal{NP}$-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 to apply the algorithm to a dynamic context, with orders arriving over time.

1. Introduction

In this paper we address a scheduling problem called *single-courier meal delivery scheduling problem* (*MDSP*) motivated by a food pick-up and delivery application in Rome, Italy. A restaurant collects food orders from the final customers. When issuing an order, the customers specify their location and the ideal delivery time. 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. This is mainly due to the fact that the quality of food decreases after a certain time from departure (Fikar & Braekers, 2022) or that, to cope with the congested traffic of big cities, couriers usually move around by bicycles or mopeds, which typically have very little space on board (Allen et al., 2021).

Here we consider the situation in which a courier can carry *one or two* orders, carefully modeling the implications of delivering two orders in the same trip vs. a single order. Moreover, our scheme can be easily adapted to forbid order pairs which are too far from each other to maintain a good level of food quality (see Section 2.2). This is a relevant difference from, e.g., parcel delivery which is carried out through vans or small trucks and which does not involve perishable goods, so that several locations can be visited during the same trip. In the following, 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 each trip. We note that in last-mile food delivery a common assumption is that couriers deliver single orders (see Seghezzi et al. (2021)), with no systematic consideration of possible aggregation among the orders. This is a different setting from the meal delivery routing problem introduced by Reyes et al. (2018) and Yildiz and Savelsbergh (2019).

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

☆ *Acknowledgments:* The authors wish to thank Ulrich Pferschy for suggesting an improvement in the complexity of Algorithm MTE.
* Corresponding author.
E-mail addresses: agnetis@diism.unisi.it (A. Agnetis), matteo.cosmi@uni.lu (M. Cosmi), gaia.nicosia@uniroma3.it (G. Nicosia), andrea.pacifici@uniroma2.it (A. Pacifici).

https://doi.org/10.1016/j.cie.2023.109514
Received 3 April 2023; Received in revised form 28 June 2023; Accepted 3 August 2023
Available online 6 August 2023

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 2.1) 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*.

In this paper we consider the 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 $MDSP_A$). The idea is that if we serve two customers in a single trip, a certain amount of travel time can be saved. However, two orders can be aggregated only if their time windows and processing times fulfill certain conditions, as explained in detail in Section 2.2 (Proposition 2). 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.

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 models on delivery problems addressed in the literature. Hence, in $MDSP_A$, scheduling decisions, rather than routing decisions, are crucial.

The paper is organized as follows. In the next subsections we first summarize the main literature results connected to our problem and then list our main contributions. In Section 2 we begin by providing a formal description of $MDSP_S$, along with the definition of the main notation used in the paper (Section 2.1). Then, in Section 2.2 we introduce the concept of order aggregation and the corresponding additional notation, give a formal definition of problem $MDSP_A$ and, in Section 2.3 characterize its complexity. In Section 3.1, we outline the description of different lower bounds that are utilized in a branch and bound algorithm, which is subsequently illustrated in Section 3.2. Section 4 presents the results of an extensive computational campaign. Finally, in Section 5 some conclusions are drawn.

1.1. Related literature

Numerous papers have addressed issues related to food delivery. Below, we present a concise overview of the key findings relevant to the addressed problems $MDSP_S$ and $MDSP_A$.

$MDSP$ 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) and Klein and Steinhardt (2023)), 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 $SDDP$, some 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 a dynamic programming-based approximation procedure to select a 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.

Reyes et al. (2018), Steever et al. (2019) and Ulmer et al. (2020) consider a stochastic, dynamic food delivery problem. Hence, they focus on developing policies, based on heuristic methods, to deal with uncertainty and to decide whether to group orders, assign them to couriers or postpone the decision. Recently, Bozanta et al. (2022) and Chen et al. (2021) propose novel machine learning based algorithms to deal with the problem of assigning orders to couriers. Liu (2019) consider a futuristic food delivery problem in which drones are used to perform delivery. Although the meal delivery problem was introduced in Reyes et al. (2018) as a stochastic dynamic problem, also its static and deterministic version has been widely studied. To solve this version of the problem, both heuristic (Joshi et al., 2021; Teng et al., 2021; Xue et al., 2021) and exact methods (Cosmi et al., 2019a, 2019b; Cosmi, Oriolo et al., 2019) have been proposed.

Albeit the literature on food delivery is relatively recent, authors have already investigated a large number of different objective functions. When orders have to be delivered as soon as possible, the click-to-door and ready-to-door time are the most used objectives. On the contrary, the number of performed deliveries and the (late) delivery costs are widely used when orders are characterized by strict delivery time windows.

Table 1 offers a comprehensive overview of the existing literature on food delivery, adopting a similar approach to other papers, like Abbasi et al. (2023). Each row of the table refers to a specific paper (whose reference appears in Column 1). Since the literature addresses various problems related to food delivery, these problems are classified based on the characteristics outlined below. Columns 2 and 3 specify whether single (S) or multiple (M) restaurants or couriers are involved. The utilized order delivery policy (as soon as possible ($ASAP$) or within predetermined time-windows (TW)) is reported in Column 4. The possibility of order aggregation is reported in Column 5, while Column 6 illustrates the considered objective function, where CtD denotes the click-to-door time, RtD the ready-to-door time, and " * " an objective function obtained weighing 10 different objectives related to three main features (safety, lateness and efficiency). Column 7 refers to whether a static (St) or dynamic (D) version of the problem is considered. The (main) proposed solution approach is reported in Column 8, while, the proposed (or not) complexity analysis is in Column 9 (the entry "$\mathcal{NP}$-*hard*" means that although a complexity analysis is not proposed, the considered model is known to be $\mathcal{NP}$-hard). Finally, the last column reports on whether the computational tests are performed on real-world data or not.

Based on Table 1, it is evident that our work stands out as the only one that conducts a formal analysis of the complexity associated with the considered food delivery problem. Furthermore, our study extends and enhances the results previously obtained in Cosmi et al. (2019b). Moreover, the findings presented in the table demonstrate that the proposed exact method is applicable to a rolling-horizon optimization approach, allowing to effectively address the dynamic version of the problem.

As we will see in detail in the next section, $MDSP_S$ can be seen as a special case of the well known single machine scheduling problem $1|r_j|\sum U_j$, in which 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 & 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

Table 1
A summary of pertaining food delivery literature.

Reference	Num. Rest.	Num. couriers	Order delivery	Order Aggr.	Objective function	Static/ dynamic	Sol. approach	Complexity analysis	Real data
Reyes et al. (2018)	*M*	*M*	*ASAP*	✓	*CtD, RtD*	*D*	*Heuristic*	✗	✓
Yildiz and Savelsbergh (2019)	*M*	*M*	*ASAP*	✓	*CtD, RtD*	*St*	*ILP*	✗	✓
Liu (2019)	*M*	*M*	*ASAP*	✓	*	*St,D*	*ILP Heuristic*	✗	✗
Steever et al. (2019)	*M*	*M*	*ASAP*	✗	*Earliness PtD, RtD*	*St,D*	*MILP Heuristic*	✗	✗
Cosmi, Oriolo et al. (2019)	*S,M*	*S,M*	*TW*	✗	*Number of ontime deliveries*	*St*	*Dynamic Programming*	✗	✗
Cosmi et al. (2019a)	*S*	*S*	*TW*	✓	*Num of deliveries*	*St*	*Lower bound*	✗	✓
Cosmi et al. (2019b)	*S*	*S*	*TW*	✓	*Num of deliveries*	*St*	*ILP*	✗	✓
Liao et al. (2020)	*M*	*M*	*TW*	✓	*Customer Satisf. emissions*	*St*	*Heuristic*	✗	✗
Ulmer et al. (2020)	*M*	*M*	*ASAP*	✓	*Sum of tardiness*	*D*	*Heuristic*	✗	✗
Chen et al. (2021)	*M*	*M*	*ASAP*	✗	*Avg disp cost*	*D*	*Heuristic*	✗	✓
Teng et al. (2021)	*M*	*M*	*TW*	✓	*Tot delivery cost*	*St*	*Heuristic*	$\mathcal{NP}$*-hard*	✓
Xue et al. (2021)	*M*	*M*	*ASAP*	✓	*Tot cost scheduling time*	*St*	*ILP Heuristic*	✗	✓
Joshi et al. (2021)	*M*	*S,M*	*ASAP*	✓	*Sum of tardiness*	*St*	*Heuristic*	✓	✓
Bozanta et al. (2022)	*M*	*S,M*	*ASAP*	✗	*Total reward*	*D*	*Heuristic*	✗	✗
Cosmi et al. (2022)	*M*	*M*	*TW*	✗	*Weighted delays*	*St,D*	*ILP*	✗	✓
This work	*S*	*S*	*TW*	✓	*Num of deliveries*	*St,D*	*Lower bound B&B*	✓	✓

Table 2
Literature related to scheduling problem $P|r_j|\sum U_j$ with fixed additive laxity.

Reference	Machines	Job aggregation	Objective	Complexity	Solution approach
Cieliebak et al. (2004)	m	✗	*Feasibility*	$O\left(n(\delta+1)^H \cdot H log(H)\right)$	*Dynamic programming*
van Bevern et al. (2017)	m	✗	*Feasibility*	$O\left(n(\delta+1)^{(2\delta+1)} \cdot (2\delta+1) \cdot log(2\delta+1)\right)$	*Dynamic programming*
Cosmi, Oriolo et al. (2019)	m	✗	$\sum U_j$	$O\left(n^m \cdot (\delta+2)^{2m(\delta+1)}\right)$	*Dynamic programming*
Böhm et al. (2021)	1	✗	$\sum w_j(1-U_j)$	$O\left(n^3 \cdot \sum w_j \cdot log(n\sum w_j)\right)$	*Dynamic programming*
This work	1	✓	$\sum U_j$	$\mathcal{NP}$*-Hard*	*Branch & Bound*

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, Oriolo et al. (2019) 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) proposed an algorithm that exactly solves this problem in $O(n^3 \cdot \sum w_j \cdot log(n\sum w_j))$. So, $MDSP_S$ is indeed polynomially solvable. We will see in this paper that this is not the case when order aggregation is allowed, that is when we consider problem $MDSP_A$.

The above results on problems connected to $1|r_j|\sum U_j$ with fixed additive laxity or fixed slack are summarized in Table 2, where each row of the table refers to a specific reference reported in the first column. The second column of the table indicates the number of considered (identical and parallel) machines, while in Column 3 is reported whether the possibility of aggregating jobs or not has been considered. Column 4 contains the addressed objective, while Column 5 illustrates the computational complexity of the problem (where H is the maximum number of overlapping time-windows, n is the number of jobs, m the number of machines, and δ the additive laxity). Finally, the last column, lists the adopted solution algorithm.

1.2. Our contribution

In this work we consider a model for scheduling single- or dual-order deliveries arising in a food pick-up and delivery application in Rome, Italy. In our study, we undertake a rigorous analysis to determine the complexity of the food delivery problem under investigation. Through this analysis, we establish that the problem, specifically when order aggregation is involved, falls into the class of $\mathcal{NP}$-hard problems. To solve the problem at hand, we first introduce novel and advanced lower bounds to provide tight estimates. Subsequently, we develop a combinatorial branch and bound algorithm, which uses the proposed lower bounds to efficiently explore the solution space and identify optimal or near-optimal solutions. To evaluate the performance of the solution algorithm, we conduct a computational study using instances derived from the considered real-life application, as well as randomly generated instances. Additionally, we compare the behavior of the combinatorial algorithm with the best-known Integer Linear Programming formulation for the problem. Furthermore, we conduct additional computational experiments to show that by appropriately selecting design parameters, the algorithm can be applied to a dynamic context where orders arrive over time.

2. Problem definition, notation and complexity

2.1. Problem $MDSP_S$

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* $J = \{1, \dots, n\}$ 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/he is not available for processing any other order until the courier 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.

2.2. Problem $MDSP_A$

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, as it 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}. An example describing the above discussion is hereafter.

Example 1. In Fig. 1 an example of the two possible alternatives for a courier that has to serve two customer orders i and j is illustrated. In particular, if the courier delivers a single order i (or j) with ideal delivery times $\hat{d}_i = 7$ ($\hat{d}_j = 10$), he/she starts at time s_i (s_j) from the restaurant R and gets back to R at time C_i (C_j). Here, $\delta = 2$ and, using definitions (2), (3), and (4), we have $r_i = 1$, $p_i = 10$, and $d_i = 13$ (respectively $r_j = 0$, $p_j = 18$, and $d_j = 20$). Note that, clearly, in this case it is impossible for the courier to serve both i and j with two distinct consecutive trips. Alternatively, the courier may aggregate the two orders: so doing, he/she starts at time s from R and then gets back to R at time C. Using definitions (5), (6), and (7), we have $p_{ij} = 18$, $r_{ij} = \max\{1, 0\} = 1$, and $d_{ij} = \min\{21, 20\} = 20$. In this example, from (8) and (9), the slack $\delta_{ij} = \delta - |\tau_{ij}| = 2 - |1| = 1$ is nonnegative and hence the twin is feasible.

For a twin ij to be feasible, the ideal delivery dates $\hat{d}_i$ and $\hat{d}_j$, as well as the travel distance t_{ij}, must satisfy certain conditions. If the locations of i and j are far from each other, then the ideal delivery times must be such that the courier has enough time to visit both of them, i.e., the difference $t_{ij} - (\hat{d}_j - \hat{d}_i)$ must not exceed the slack δ. Similarly, if the ideal delivery times $\hat{d}_i$ and $\hat{d}_j$ are relatively close to each other, t_{ij} must not be too large, as the courier would not be on time to move from i to j. Moreover, if $\hat{d}_j - \hat{d}_i$ is large, the locations of i and j cannot be too close, since this would force the courier to wait, which is forbidden by the no-wait assumption. The following proposition, introduced in Cosmi et al. (2019b), formally establishes these conditions in a unified way. Here we report a novel proof for this result.

Proposition 2. *Given two orders $i, j \in J \times J$, the twin ij is feasible if and only if:*

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

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 $\tau_{ij} \stackrel{\text{def}}{=} t_{ij} - (d_j - d_i) + \frac{1}{2}(p_j - p_i)$. 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:

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

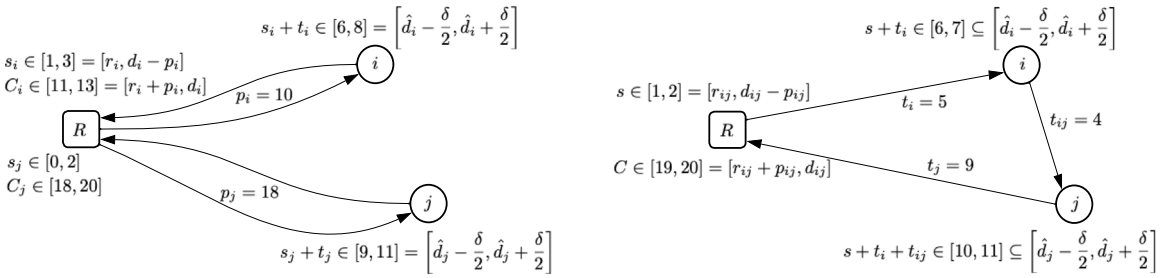


Fig. 1. Alternative courier trips.

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

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:

$$r_{ij} + t_i + t_{ij} + t_j \leq d_{ij}. \tag{12}$$

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

$$\delta_{ij} = \delta - |\tau_{ij}| \geq 0 \tag{13}$$

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

$$d_{ij} = r_{ij} + p_{ij} + \delta_{ij}. \tag{14}$$

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

$$D = \{ij \ : \ (i,j) \in J \times J, \ \delta_{ij} \geq 0\}. \tag{15}$$

We note here that our model can be easily generalized if one wants to restrict the set of feasible twins. In particular, one may argue that even if i and j are such that $\delta_{ij} \geq 0$, the two locations may be too far apart to guarantee the quality of food when it reaches j. This only requires to define D as $\{ij \ : \ (i,j) \in J \times J, \ \delta_{ij} \geq 0, t_{ij} \leq T_{max}\}$, where $T_{\max}$ is the maximum travel time allowed between two orders. For simplicity of exposition, we will not consider $T_{\max}$ in the following.

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. Note that it is not necessary to consider twins in which one order is on time and the other one is late. In fact, if a schedule σ exists in which there is a twin ij with i on time and j late, then consider another schedule $\bar{\sigma}$ identical to σ except that i and j are split into two single orders. Then only i is on time and $f(\bar{\sigma}) \leq f(\sigma)$ since in $\bar{\sigma}$ the courier is back to restaurant earlier than in σ. 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 times $\hat{d}_j$, $j = 1, \ldots, n$, the travel times t_j between the restaurant and customer j, $j = 1, \ldots, n$, the travel times 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 $j \in J$).

2.3. Complexity of $MDSP_A$

While problem $MDSP_S$ is solvable in polynomial time (see Section 1.1 and Table 2), hereafter 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 $\sum_{i \in S} a_i = \frac{1}{2} \sum_{i=1}^{q} a_i$?

Observation 3. *Note that* PARTITION *remains $\mathcal{NP}$-complete even if all the a_i are even numbers. In fact, given any instance of* PARTITION*, if we double all integers we obtain another instance which is a YES-instance if and only if the original instance is a YES-instance.*

Theorem 4. *Problem $MDSP_A$ is $\mathcal{NP}$-hard.*

Proof. Consider an instance I of PARTITION with q integers $a_1, a_2, \ldots, a_q$. In view of Observation 3 above, we assume that all values a_i are even. 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 $d_i = d_{i'} = 2W + 1$, $i, i' = 1, \ldots, q$. Note that because of (3), the values p_i must be even, and this is ensured by the initial assumption on values a_i. We refer to these $2q$ 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 3.

Clearly, in I' we have $\delta = W + 1$. In addition, the travel time between two partner orders i, i' is set to zero, $i, i' = 1, \ldots, q+2$. For any other pair of orders, 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

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

Table 3
Data of instance I'.

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

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:

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

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 Fig. 2.

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. □

3. Methodologies

In this section we present our solution approach to $MDSP_A$, based on an *ad-hoc* branch and bound, whose performance falls back on the quality of lower bounds provided during the enumeration procedure. In the next sections, we give details about how these bounds can be efficiently computed.

3.1. Lower bounds

In Cosmi et al. (2019a), combinatorial lower bounds are presented that 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 upon 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 $1 \parallel \sum U_j$ to the problem with agreeable release and due dates.

In the following, we describe how the auxiliary instance P_{aux} is defined. Recall that there is a job in P_{aux} for each order in the original instance of $MDSP_A$ and viceversa.

Given an instance of $MDSP_A$, we first illustrate how release dates and due dates are defined for the jobs in the corresponding instance of P_{aux}, and then we give two possible definitions of the jobs processing times.

3.1.1. Definition of time windows in P_{aux}

We next illustrate how to compute release dates and due dates in an instance of P_{aux}. 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, $j = 1, \dots, n$, 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

$$\tilde{I}_j = \bigcup_{i:\, I_i \supset I_j} I_i \tag{16}$$

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 $\tilde{d}_j$).

Theorem 5. *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 (16), no interval $\tilde{I}_j$ is affected, for any $j \neq i$. In fact, for this to happen, one should have that $I_j \subset I'_i$

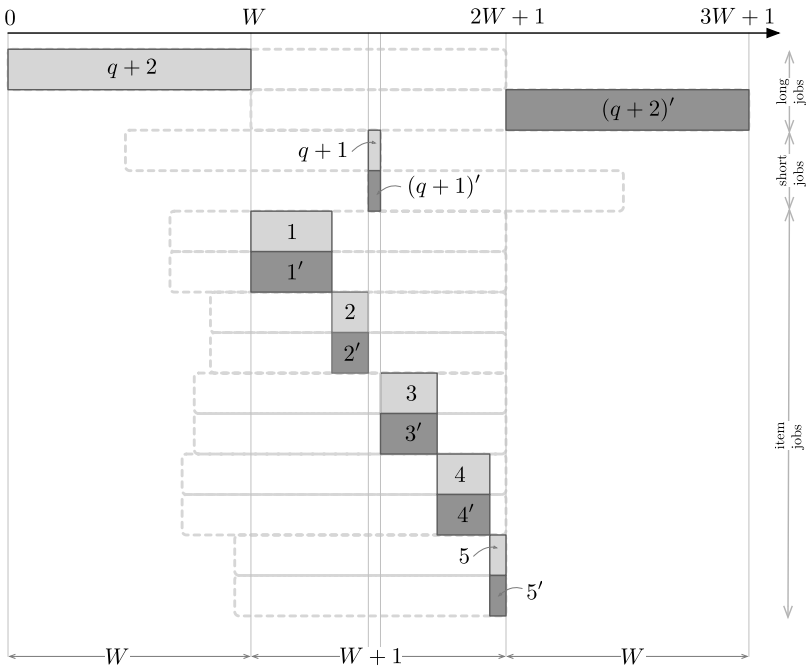


Fig. 2. Illustration of a schedule corresponding to a YES-instance of Partition with five items. Dotted rectangles indicate the orders windows $[r_i, d_i]$.

Algorithm 1 Minimum Total Enlargement (MTE) Algorithm

Input: Intervals $[r_j, d_j] \subset \mathbb{R}$ for all orders $j \in J$
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**
    $r'_j := \tilde{r}_j$; $d'_j = d_j$;
  else
    $r'_j = r_j$; $d'_j := \tilde{d}_j$;
  end if
end for
Return intervals $I'_j = [r'_j, d'_j] \subset \mathbb{R}$ for all orders $j \in J$

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. □

3.1.2. Complexity of Algorithm 1

A straightforward computation of the intervals $\tilde{I}_j$ requires time $O(n^2)$. A more careful implementation that may improve the complexity of the above procedure is sketched hereafter. For every job j, consider the set

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

(Recall that if $d_i = d_j \vee r_i = r_j$ then i and j are agreeable.) Clearly, if $Q_j = \emptyset$, 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 $h \in J$ whose intervals are not strictly contained in any other interval, that is, no other job $\ell \in J$ exists with $r_\ell < r_h$ and $d_\ell > d_h$, i.e., $Q_h = \emptyset$. $\tilde{J}$ can be found in $O(n \log n)$ time by simply sweeping through the ordered set of jobs. Note that $\tilde{J} = \{h(1), h(2), \dots, h(q)\}$ with $r_{h(\ell)} \leq r_{h(\ell+1)}$ and $d_{h(\ell)} \leq d_{h(\ell+1)}$, $\ell = 1, \dots, q-1$.

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

Let $\tilde{d}_0 = -\infty$. If $d_j \geq \tilde{d}_{j-1}$ then, since $r_j \geq \tilde{r}_{j-1}$, jobs j and $j-1$ are agreeable and we just let $\tilde{d}_j := d_j$. On the other hand, if $d_j < \tilde{d}_{j-1}$, an enlargement is needed and $\tilde{d}_j := \tilde{d}_{j-1}$.

As for the leftward enlargement $\tilde{r}_j$, we have to determine the job in Q_j (the set of jobs whose intervals strictly contain I_j) with minimum release date. Then $\tilde{r}_j = \min\{r_i \ : \ d_i > d_j, i \in \tilde{J}\}$. This, in turn, corresponds to searching for the job $h \in \tilde{J}$ with minimum due date larger than d_j, which can be done in $O(\log n)$ time through a binary search. 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 Algorithm 1.

3.1.3. Definition of processing times in P_{aux}

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 $MDSP_A$.

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. To this purpose, we consider two different ways of defining job processing times p'_j in the auxiliary instance P_{aux}.

1. *Flat processing times*. We set the duration of job j equal to

$$p'_j = t_j + \min\left\{t_j, \min_k\left\{\frac{1}{2}t_{jk}\right\}\right\}. \tag{17}$$

2. *Order-based processing times*. We renumber all jobs in P_{aux} so that $i < j$ if $d'_i < d'_j$ or $d'_i = d'_j$ and $r'_i < r'_j$. Then the processing times p'_j are defined as follows:

$$p'_j = \begin{cases} p_j & j = 1 \\ \min\{p_j, \min\limits_{\substack{h<j: \\ (h,j)\in D}}\{p_{hj} - p'_h\}, \min\limits_{\substack{h<j: \\ (j,h)\in D}}\{p_{jh} - p'_h\}\} & j > 1. \end{cases} \tag{18}$$

Hereafter, we show that – for both definitions of processing times – the optimal solution value of P_{aux} always provides a lower bound for

$MDSP_A$.

3.1.4. First lower bound: P_{aux} with flat processing times

First consider flat processing times as in (17).

Theorem 6. *Each feasible schedule σ of value z for the original problem $MDSP_A$ defines a sequence of jobs which corresponds to a feasible schedule σ' of value z' for problem P_{aux} with flat processing times, and $z' \le z$.*

Proof. From (17), it is easy to observe that $p'_j \le p_j$. Moreover, for 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 sequence σ of orders, solution of $MDSP_A$ with z tardy orders, if we sequence the jobs in P_{aux} in the same order as in σ, we get a schedule with $z' \le z$ tardy jobs. □

3.1.5. Second lower bound: P_{aux} with order-based processing times

Consider now P_{aux} with order-based processing times as in (18). Given an order j, we use the notation $m(j)$ to indicate the order such that:

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

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

Lemma 7. *In problem P_{aux} with order-based processing times, for each job j, the following inequalities hold: (i) $p'_j \le p_j$ and (ii) $p'_j + p'_{m(j)} \le p_{j,m(j)}$.*

Proof. From (18), it easily follows that $p'_j \le p_j$.

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

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

From (18), we have that $p'_j \le p_{hj} - p'_h$. Hence $p'_j + p'_h \le p_{hj}$.

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

2.1 If $m(j) = l > j$ and p'_l is attained in correspondence to a certain index j, then Eq. (18) gives:

$$p'_l = \min\{p_l, p_{jl} - p'_j, p_{lj} - p'_j\}.$$

Hence, $p'_l \le p_{jl} - p'_j$, so again $p_{jl} \ge p'_l + p'_j$.

2.2 If $m(j) = l > j$ and p'_l is attained in correspondence to a certain index $k \ne j$, then Eq. (18) gives:

$$p'_l = \min\{p_l, p_{kl} - p'_k, p_{lk} - p'_k\}.$$

Hence, $p'_l \le p_{kl} - p'_k \le p_{jl} - p'_j$, so again $p_{jl} \ge p'_l + p'_j$.

This completes the proof. □

Theorem 8. *Each feasible schedule σ of value z for the original problem $MDSP_A$ defines a sequence of jobs which corresponds to a feasible schedule σ' of value z' for problem P_{aux} with order-based processing times, and $z' \le z$.*

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 (18);
- 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 $s'_j \ge r'_j \ge r_j$.

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 $d'_j \ge d_j$ for each job $j \in J$.

1. *Single-order trips.* Consider a job j corresponding to an order j which is served in a single-order trip and it is on time in σ. Then, in σ', job j completes at $C'_j = s'_j + p'_j = s_j + p'_j \le s_j + p_j \le d_j \le d'_j$, hence j is on time also in σ'.
2. *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:
   (a) *First Job.* The first job i starts at $s'_i = s_i$ and it ends at $C'_i = s_i + p'_i \le s_i + p_i \le d_i \le d'_i$, hence i is on time in σ'.
   (b) *Second Job.* In σ', job j starts either at its release time $r'_j \le r_j$ or when the job i ends at C'_i. In the first case, since obviously $r_j + p_j \le d_j$, and since $r'_j \le r_j$, $p'_j \le p_j$ and $d'_j \ge d_j$, one has $r'_j + p'_j \le r_j + p_j \le d_j \le d'_j$. In the latter case, $C'_j = s_i + p'_i + p'_j$. From Lemma 7, we have $p'_i \le p_{i,m(i)} - p'_{m(i)} \le p_{ij} - p'_j$ and hence $p'_i + p'_j \le p_{ij}$, so that $C'_j \le s_i + p_{ij}$. Since the twin is on time in σ, one has $s_i + p_{ij} \le d_j$ and, in conclusion, $C'_j \le d'_j$, i.e., also in this case job j is on time in σ'.

Since each job j which is on time in σ for $MDSP_A$ corresponds to a job on time in σ' for P_{aux}, we have that $z' \le z$. This completes the proof. □

The following example illustrates the above procedure for computing a lower bound through the definition of a suitable instance of P_{aux}.

Example 9. 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 Eqs. (5), (6), and (7) – are as follows:

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

$$\{d_{ij}\} = \begin{pmatrix} 40 & 41 & - & 50 & - \\ 40 & 41 & - & - & - \\ - & - & 47 & - & 47 \\ 40 & - & - & 50 & - \\ - & - & 47 & - & 47 \end{pmatrix}$$

According to MTE Algorithm 1 and Eq. (18) we compute release times (r'_j), due dates (d'_j) and order-based 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 (Kise et al., 1978) 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 $\sigma = \langle$"12","35"$\rangle$ in which job 4 is late.

3.2. A combinatorial branch and bound

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

B&B works as follows. We adopt a n-ary branching, which is standard for sequencing problems (see Pinedo (2012)): In the enumeration tree, each node l is associated to a partial sequence σ_l of, say, $n(l)$ jobs. The root node corresponds to an empty sequence. So, at node l there

is a set J_l of $n - n(l)$ jobs whose position in the sequence has not yet been determined. For each unscheduled job $j \in J_l$ we consider a child node: Each of these nodes is associated to a partial sequence $\langle \sigma_l, j \rangle$ with $n(l) + 1$ jobs in which j is appended to σ_l, i.e., j is the last job in the partial sequence. Moreover, the last job j may either be a single-order trip or form a twin with the second-last order. Summarizing, 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 Fig. 3, 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, in which k is the last scheduled order, and consider 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 :
  - $\nexists j \in J_l : \max\{r_j, s_k + p_k\} + p_j \le r_h$
  - $\nexists j \in J_l : \max\{r_{kj}, s_k\} + p_{kj} \le r_h$
  - if h is scheduled as a single-order trip, $s_k + p_k + p_h \le d_h$
  - if h is scheduled as the second order of the twin $kh \in D$, $\max\{s_k, r_{kh}\} + p_{kh} \le d_{kh}$.
- In case (ii) above, i.e., when σ_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:
  - $\nexists j \in J_l : \max\{s_{ik} + p_{ik}, r_j\} + p_j \le r_h$.

Lower Bounds. At each node l of the enumeration tree, a lower bound is computed solving P_{aux} with flat and order-based processing times, and then selecting the largest among them. If such lower bound is not smaller than the incumbent best solution, the node is fathomed. The lower bound with order-based processing times is computed applying Algorithm 2. The lower bound with flat processing times is computed through the same algorithm, except that lines 12–16 are simply replaced by the application of (17).

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, and we 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 we let t^* be the completion time of the order that immediately precedes k in σ_l, which is the first available starting time for the other trips in this case. 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.

Algorithm 2 Twin Lower Bound

1: **Input:** Partial schedule σ_l
2: **if** the last job k in σ_l is a single order-trip **then**
3: $\quad \hat{J}_l = J_l \cup \{k\}$
4: $\quad t^* :=$ completion time of the order that immediately precedes k in σ_l,
5: **else**
6: $\quad \hat{J}_l = J_l$
7: $\quad t^* :=$ completion time of σ_l
8: **end if**
/* Build problem P_{aux} at l: */
9: $\quad$ Use MTE Algorithm on the job set $\hat{J}_l$ to compute release times and due dates
10: $\quad$ Build EDD sequence of jobs in $\hat{J}_l$ and rename jobs according to it
11: $\quad$ **for** $j \in \hat{J}_l$ **do**
12: $\quad\quad$ **if** $j = 1$ **then**
13: $\quad\quad\quad p'_j = p_j$
14: $\quad\quad$ **else** $j \neq 1$
15: $\quad\quad\quad p'_j = \min\{p_j, \min_{\substack{h<j: \\ (h,j)\in D}} \{p_{hj} - p'_h\}, \min_{\substack{h<j: \\ (j,h)\in D}} \{p_{jh} - p'_h\}\}$
16: $\quad\quad$ **end if**
17: $\quad$ **end for**
/* Solve P_{aux}: */
18: $\quad$ Run KIM algorithm (Kise et al., 1978) from time t^*
19: $\quad$ **Return** the minimum number z_l of late orders computed for problem P_{aux} at node l

The set of orders that Algorithm 2 should consider, depending on l, is referred to as $\hat{J}_l$ and the first available time for those orders is indicated by t^*. 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 this 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 $C_j \le d_j$. 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 $C_{kj} \le d_{kj}$, we let $w_{kj} = C_{kj} - t^*$. Finally, if a twin ij is appended to k, completing at $C_{ij} \le d_{ij}$, 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 is provided in order to give an idea of our *ad-hoc* branching procedure.

Example 10. 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

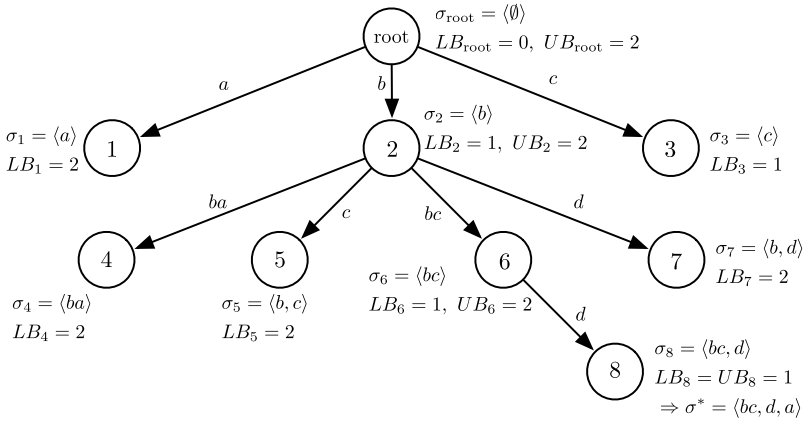


Fig. 3. An example of the enumeration tree of the branch and bound procedure.

in Fig. 3. 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 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 $UB_2 = 2$ is computed. 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.

4. Computational experiments

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 3.2 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 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, 2018) in its multithread version. The time limit is set to one hour, for both B&B and Gurobi.

4.1. 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 h, 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 delivery times. The distributions of ideal delivery times around lunch time and dinner time are shown in Fig. 4(a) and (b) respectively.

In Table 4, for each n between 5 and 31, the number of instances with n orders is reported.

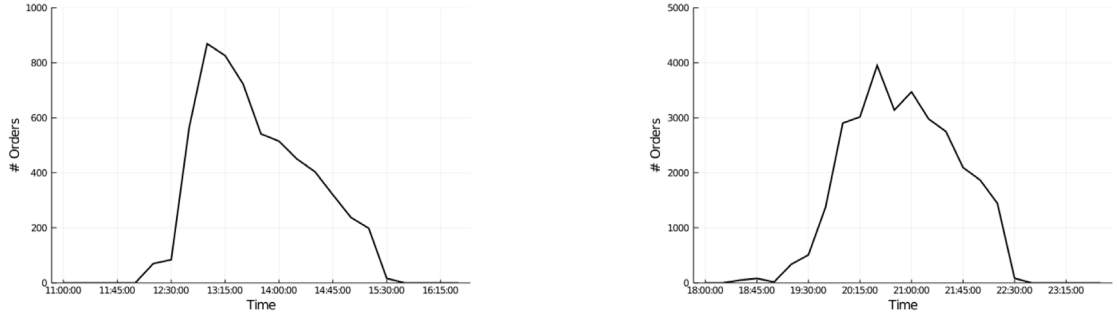
The delivery time window of each order i is centered around the ideal delivery time $\hat{d}_i$, and the slack is $\delta = 30$ min. This corresponds to assuming that a delivery is acceptable as long as its earliness or tardiness with respect to $\hat{d}_i$ does not exceed 15 min, which corresponds to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

While the analysis of the effectiveness of our solution approach is presented in Section 4.1.2, we next comment on the values of the number of late order with and without order aggregation.

4.1.1. 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 Appendix.) The results are summarized in Fig. 5, in which, for each value of n, the average number of late deliveries is displayed with and without order aggregation.

From Fig. 5 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



(a) Lunch (b) Dinner

Fig. 4. Distribution of ideal delivery times in real-life instances.

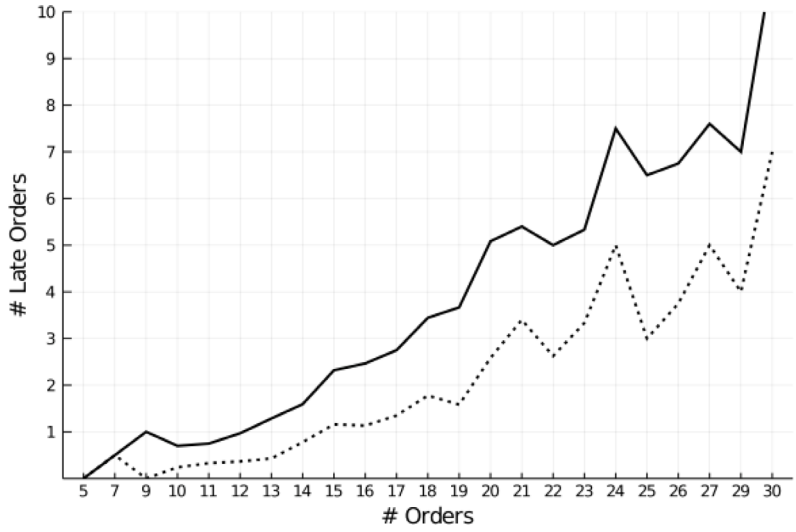


Fig. 5. Average number of late deliveries in real-life instances when order aggregation is allowed (dashed line) and when it is not (solid line).

that in 212 out of 476 optimally solved instances, order aggregation reduces the number of late deliveries (compared to the single-order setting). 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.

4.1.2. 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 4. 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 h 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 s, hence on the average B&B is 8 times faster than Gurobi on these instances.

4.2. 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. 4(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

Table 4
Results of B&B vs. ILP for real-life instances.

n	# Inst	B&B solved	ILP solved	B&B CPU s	ILP CPU s
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[a]
17	20	20	20	23.90	124.32
18	18	18	16	39.06	82.13[a]
19	12	12	11	13.46	307.38[a]
20	12	12	8	92.76	326.89[a]
21	5	5	2	6.30	908.44[a]
22	8	8	5	8.07	490.32[a]
23	3	3	2	291.02	2743.12[a]
24	2	2	0	483.53	–
25	2	2	0	390.77	–
26	4	4	1	612.17	1590.40[a]
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*

[a]CPU time refers to solved instances only.

Table 5
Results of B&B for Gaussian instances.

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

[a]CPU time refers to solved instances only.
[b]The B&B gaps refer to unsolved instances only.

Table 6
Results of B&B for Uniform instances.

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

[a]CPU time refers to solved instances only.
[b]The B&B gaps refer to unsolved instances only.

Table 7
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 8
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

the processing times p_j and travel times t_{ij}. The results are shown in Tables 5–8.

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 s for Uniform instances, 876.1 s for Gaussian instances) than in the real-life instances (11.18 s). In other words, B&B appears more suitable to deal with congested instances than the ILP.
- Tables 7 and 8 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.

4.3. 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

Table 9
Average number of late orders in the 478 real-life instances for various values of T.

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

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., orders 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 9. Each row shows the average value of the number of late orders on the whole set of 478 real-life instances.

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.

5. 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 either 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.

Future research may address overcoming some possible limitations of the current model. This includes the following issues: (*i*) $MDSP_A$ is a deterministic model, while some aspects of the problem (e.g., travel times and/or waiting times at the restaurant) might be addressed by a robust optimization or stochastic model (Bozanta et al., 2022; Liu, 2019; Teng et al., 2021). 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*) In a more general setting, the same courier may be shared by multiple restaurants. This would bring 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) and Nicosia et al. (2018)). (*iii*) Another issue concerns the no-wait assumption (see Section 2.2), which might not hold in general. For instance, some limited slack time between two deliveries in the same trip may be allowed, or the twin is considered feasible if the second order is delivered within a certain time from the departure from the restaurant.

Finally, from the viewpoint of problem analysis, we point out that the complexity of $MDSP_A$ is still open as for strong $\mathcal{NP}$-hardness. Moreover, a branch and price ad-hoc algorithm may be pursued using either a time-indexed formulation or a packing formulation (see, e.g., Agnetis et al. (2009)).

CRediT authorship contribution statement

Alessandro Agnetis: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization, Supervision. **Matteo Cosmi:** Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization, Supervision. **Gaia Nicosia:** Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization, Supervision. **Andrea Pacifici:** Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization, Supervision.

Data availability

Data will be made available on request.

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 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 $i = j$.

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.

$$\min \sum_{ij \in D} w_{ij} y_{ij} \tag{19}$$

$$s.t. \quad s_i \geq \sum_{ij \in D} r_{ij} \beta_{ij} \qquad i \in J \tag{20}$$

$$s_j \geq c_i - M(\beta_{ij} - x_{ij} + 1) \qquad i,j \in J : i \neq j \tag{21}$$

$$s_i \geq c_j - M(\beta_{ij} + x_{ij}) \qquad i,j \in J : i \neq j \tag{22}$$

$$s_j \geq s_i - M(1 - \beta_{ij}) \qquad ij \in D \tag{23}$$

$$s_i + p_{ij} \leq d_{ij} + M(y_{ij} + 1 - \beta_{i,j}) \qquad ij \in D \tag{24}$$

$$s_j + p_{ij} \leq d_{ij} + M(y_{ij} + 1 - \beta_{i,j}) \qquad ij \in D \tag{25}$$

$$c_j = s_j + \sum_{i : ij \in D} p_{ij} \beta_{ij} \qquad j \in J \tag{26}$$

$$x_{ij} + x_{ji} = 1 \qquad ij : i \neq j \in D \tag{27}$$

$$x_{jh} \geq x_{ih} - M(1 - \beta_{ij}) \qquad \begin{matrix} ij \in D \\ h \in J : h \neq i,j \end{matrix} \tag{28}$$

$$\sum_{j : ij \in D} \beta_{ij} \leq 1 \qquad i \in J \tag{29}$$

$$\sum_{i:ij\in D} \beta_{ij} \le 1 \qquad j \in J \qquad (30)$$

$$\sum_{j:ij\in D} \beta_{ij} + \sum_{h:hi\in D, h\neq i} \beta_{hi} = 1 \qquad i \in J \qquad (31)$$

$$s_i, c_i \in \mathbb{R}_+ \qquad i \in J \qquad (32)$$

$$\beta_{ij}, x_{ij}, y_{ij} \in \{0,1\} \qquad ij \in D \qquad (33)$$

Constraints (29)–(31) impose that each order i must always be associated and hence scheduled within a twin order (including order (i,i)). Constraints (20) force each order i to always start after the release time of the twin order it is associated to. Constraints (21)–(22) 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 (23) holds. Each single order has to be completed before its twin order delivery time otherwise the twin order is considered late (24)–(25). Constraints (27) impose that if i precedes j then it is not possible that j precedes i. Constraint (28) sets that if i and j are scheduled in the twin order ij then if i precedes h also j has to precede h.

References

Abbasi, S., Daneshmand-Mehr, M., & Ghane Kanafi, A. (2023). Green closed-loop supply chain network design during the coronavirus (COVID-19) pandemic: a case study in the Iranian automotive industry. *Environmental Modeling & Assessment*, *28*, 69–103.

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

Agnetis, A., Nicosia, G., Pacifici, A., & Pferschy, U. (2013). Two agents competing for a shared machine. In P. Perny, M. Pirlot, & A. Tsoukiàs (Eds.), *Algorithmic decision theory* (pp. 1–14). Berlin, Heidelberg: Springer Berlin Heidelberg.

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

Allen, J., Piecyk, M., Cherrett, T., Juhari, M. N., McLeod, F., Piotrowska, M., Bates, O., Bektas, T., Cheliotis, K., Friday, A., & Wise, S. (2021). Understanding the transport and CO2 impacts of on-demand meal deliveries: A London case study. *Cities*, *108*, Article 102973.

Aloulou, M. A., & 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., & Speranza, M. G. (2015). Complexity of routing problems with release dates. *European Journal of Operational Research*, *247*(3), 797–803.

Azi, N., Gendreau, M., & 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., & Shah, V. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, *59*(1), 65–98.

Böhm, M., Megow, N., & Schlöter, J. (2021). Throughput scheduling with equal additive laxity. In T. Calamoneri, & F. Corò (Eds.), *Algorithms and complexity* (pp. 130–143). Springer International Publishing.

Bozanta, A., Cevik, M., Kavaklioglu, C., Kavuk, E. M., Tosun, A., Sonuc, S. B., Duranel, A., & Basar, A. (2022). Courier routing and assignment for food delivery service using reinforcement learning. *Computers & Industrial Engineering*, *164*, Article 107871.

Chen, J.-f., Wang, L., Wang, S., Wang, X., & Ren, H. (2021). An effective matching algorithm with adaptive tie-breaking strategy for online food delivery problem. In *Complex & intelligent systems* (pp. 1–22). Springer.

Cieliebak, M., Erlebach, T., Hennecke, F., Weber, B., & Widmayer, P. (2004). Scheduling with release times and deadlines on a minimum number of machines. In J.-J. Levy, E. W. Mayr, & J. C. Mitchell (Eds.), *Exploring new frontiers of theoretical informatics* (pp. 209–222). Boston, MA: Springer US.

Cosmi, M., Nicosia, G., & 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, vol. 2019* (pp. 33–36).

Cosmi, M., Nicosia, G., & Pacifici, A. (2019b). Scheduling for last-mile meal-delivery processes. *IFACPapersOnLine*, *52*, 511–516.

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

Cosmi, M., Oriolo, G., Piccialli, V., & Ventura, P. (2022). *Assigning orders to couriers in meal delivery via integer programming*. Optimization-Online.

Detti, P., Nicosia, G., Pacifici, A., & 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., & Lubin, M. (2017). Jump: A modeling language for mathematical optimization. *SIAM Review*, *59*(2), 295–320.

Fikar, C., & Braekers, K. (2022). Bi-objective optimization of e-grocery deliveries considering food quality losses. *Computers & Industrial Engineering*, *163*, Article 107848.

Garey, M. R., & 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.

Joshi, M., Singh, A., Ranu, S., Bagchi, A., Karia, P., & Kala, P. (2021). Batching and matching for food de- livery in dynamic road networks. In *2021 IEEE 37th international conference on data engineering (ICDE)* (pp. 2099–2104). Los Alamitos, CA, USA: IEEE Computer Society.

Kise, H., Ibaraki, T., & 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., & 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., & Toriello, A. (2018b). The one-dimensional dynamic dispatch waves problem. *Transportation Science*, *52*(2), 402–415.

Klein, V., & Steinhardt, C. (2023). Dynamic demand management and online tour planning for same-day delivery. *European Journal of Operational Research*, *307*(2), 860–886.

Liao, Wenzhu, Zhang, Liuyang, & Wei, Zhenzhen (2020). Multi-objective green meal delivery routing problem based on a two-stage solution strat- egy. *Journal of Cleaner Production*, [ISSN: 0959-6526] *258*, Article 120627.

Liu, Y. (2019). An optimization-driven dynamic vehicle routing algorithm for on-demand meal delivery using drones. *Computers & Operations Research*, *111*, 1–20.

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

Pinedo, M. L. (2012). *Scheduling : Theory, algorithms, and systems*. New York, NY: Springer.

Reyes, D., Erera, A., Savelsbergh, M., Sahasrabudhe, S., & O'Neil, R. (2018). *The meal delivery routing problem*. Optimization-Online.

Seghezzi, A., Winkenbach, M., & Mangiaracina, R. (2021). On-demand food delivery: a systematic literature review. *International Journal of Logistics Management*, *32*(4), 1334–1355.

Steever, Zachary, Karwan, Mark, & Murray, Chase (2019). Dynamic courier routing for a food delivery service. *Computers & Operations Research*, [ISSN: 0305-0548] *107*, 173–188.

Teng, R., Hong-bo, X., Kang-ning, J., Tian-yu, L., Ling, W., & Li-ning, X. (2021). Optimisation of takeaway delivery routes considering the mutual satisfactions of merchants and customers. *Computers & Industrial Engineering*, *162*, Article 107728.

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

Ulmer, M. W., Thomas, B. W., & 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., & Suchý, O. (2017). A parameterized complexity view on nonpreemptively scheduling interval-constrained jobs: few machines, small looseness, and small slack. *Journal of Scheduling*, *20*(3), 255–265.

Xue, Guiqin, Wang, Zheng, & Wang, Guan (2021). Optimization of rider scheduling for a food delivery service in O2O business. *Journal of Advanced Transportation*, *2021*.

Yildiz, B., & Savelsbergh, M. (2019). Provably high-quality solutions for the meal delivery routing problem. *Transportation Science*, *53*(5), 1372–1388.