Omega 133 (2025) 103237

Contents lists available at ScienceDirect

Omega

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

Courier assignment in meal delivery via integer programming: A case study in Rome☆

Matteo Cosmi [a], Gianpaolo Oriolo [b], Veronica Piccialli [c,*], Paolo Ventura [d,e]

[a] Luxembourg Centre for Logistics and Supply Chain Management (LCL), University of Luxembourg, 6, rue Richard Coudenhove-Kalergi, L-1359 Luxembourg, Luxembourg
[b] Dipartimento di Ingegneria Civile e Ingegneria Informatica, Università degli Studi di Roma "Tor Vergata", via del Politecnico 1, 00133 Roma, Italy
[c] Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Università degli Studi di Roma "La Sapienza", via Ariosto 25, 00185, Roma, Italy
[d] Istituto di Analisi dei Sistemi ed Informatica "A. Ruberti" del CNR, via dei Taurini 19, 00185, Roma, Italy
[e] Center of Competence for Optimization, Siemens Mobility, viale Marco Polo 59, 00154, Roma, Italy

ARTICLE INFO

Dataset link: https://github.com/MatteoCosmi/Rome-Meal-Delivery-Instances

Keywords:
Integer linear programming
Deterministic demand
Allocation
Meal delivery

ABSTRACT

We present an optimization model for assigning orders to couriers developed for an Italian meal delivery firm focusing on Rome. The firm focuses on top-end restaurants and customers and pursues high Quality of Service through careful management of delays. Our model reflects that in the firm's business, the majority of orders are placed in advance. This took us to design a sequential decision process implementing a rolling horizon approach where we do not try to anticipate future demands. We, therefore, iterate the solution of a fully deterministic optimization problem, the *Offline Couriers Assignment Problem* (OCAP), where we assume full knowledge of the orders and aim at minimizing delays and rejections. We solve OCAP through integer linear programming and in particular by a "flow-like" formulation on a suitable network whose size is kept as small as possible. We validate both the quality of this formulation and the sequential decision process through some computational tests on real instances collected on the ground. We make these instances available to the scientific community.

1. Introduction

In the last few years, the spreading of e-commerce has also been reflected in the exponential growth of the meal delivery sector as represented by the spreading of companies like Deliveroo, Glovo, Just-Eat Takeaway, DoorDash, and Uber Eats. This increasing interest for the meal delivery logistics is also witnessed by several papers published in the last years in optimization journals.

One key issue for the success of many companies in the meal delivery business is that of entrusting independent contractors (i.e., couriers) for deliveries. This is the case of an Italian meal delivery company, that in the following we refer to as $\mathcal{M}$, mainly operating in the area of Rome: this work indeed stems from a collaboration with this company. $\mathcal{M}$ focuses on top-end customers and restaurants and pursues high Quality of Service (QoS), in particular in terms of delays in delivering the orders while obtaining positive commitment of couriers. To this aim, couriers are hired on a regular base, and only a part of their salary depends indeed on the orders that they deliver during the work shift. Also, the assignment of orders to couriers is centralized, and couriers cannot refuse an order that has been assigned to them, i.e., according to $\mathcal{M}$ business model, orders are therefore assigned to couriers on a "push" base, whereas other models are "pull" based and couriers will choose orders. Moreover, while a courier may choose at what time to start the work shift, once she starts, she must be available until its end. As for the management of delays, in order to prevent unpleasant confrontations between couriers and customers, orders with a delay larger than 60 min are rejected (and therefore not delivered), and $\mathcal{M}$ gives a discount voucher to customers who experience a delay, or a rejection.

Given these premises on the business model of $\mathcal{M}$, the target of our collaboration with $\mathcal{M}$ was the design of an optimization model for dispatching orders to couriers to avoid as much as possible delays and rejections. Note that this model does not take into account any cost related to the number of couriers, that are already hired, and to the cost of the routing, as each courier will autonomously choose her route. Also, orders are not aggregated and each courier takes a single order for each ride so that the quality of (fine) food does not degrade.

☆ Area: Production Management, Scheduling and Logistics. This manuscript was processed by Associate Editor Kis.

* Corresponding author.
E-mail addresses: matteo.cosmi@uni.lu (M. Cosmi), oriolo@disp.uniroma2.it (G. Oriolo), veronica.piccialli@uniroma1.it (V. Piccialli), paolo.ventura@iasi.cnr.it, paolo.ventura@siemens.com (P. Ventura).

https://doi.org/10.1016/j.omega.2024.103237
Received 26 March 2024; Accepted 22 November 2024
Available online 2 December 2024

We propose a sequential decision process solved by means of a *rolling horizon* solution strategy where at each solution stage we solve a deterministic optimization problem that assumes a complete knowledge of the orders to be served; the set of orders to be served is then updated for the next solution stage as to follow the insurgence of new demands. Our solution strategy is hence fully myopic, and we do not try to anticipate future demands.

At the core of our approach, there is the solution of a deterministic optimization problem that we call the *Offline Couriers Assignment Problem* (OCAP). It follows from Cosmi et al. [1] that OCAP is NP-HARD already for the case of a single courier. We, therefore, propose for its solution a "flow-like" Integer Linear Programming (ILP) formulation that exploits a suitable time-extended network and the structure of the problem: in particular, the hypothesis that once a courier starts her work shift, she cannot refuse any order and, in a way, cannot be "distinguished" from other active couriers. We keep under control the size of the integer linear program by exploiting the monotonicity property of the cost function for OCAP and through a careful construction of the time-extended network.

We devote the last part of the paper to some computational experiments aiming at validating our solution strategy. We deal with a set of real instances, collected on the ground in the period ranging from January to March 2019, that we make available to the scientific community. First, we show that the flow-like formulation performs quite well, both in terms of computational time and integrality gap, even when we deal with more challenging instances where we artificially reduce the number of couriers. Then, in order to validate the sequential decision process, we deal again with the same instances but in the dynamic setting described above, where we solve a sequence of instances of OCAP. Our results show that our solution to OCAP scales pretty well to this setting where the computational time available for the single instance of OCAP is bounded (namely 5 min). They also show the very good quality of the overall solution found by the sequential decision process, as its cost is always not too far away from the *lower bound* provided by the cost of the optimal solution to OCAP that assumes, for the same set of orders, full knowledge *in advance*.

Contributions of this paper. The focus of this paper is on the design and the solution of an optimization model for a specific case study. Therefore, even if we present a few facts about the business model of $\mathcal{M}$ and discuss relationships between our optimization model and other models from the literature, our main interest lies in the design of a careful optimization model that is consistent with the business model of $\mathcal{M}$ and in the design of an effective solution strategy for the model to be validated on real instances collected on the ground. At the core of this strategy, there is the definition of an optimization problem, OCAP. We design, for the solution of OCAP, an effective flow-like ILP formulation that carefully exploits some hypothesis in the business model and reduces the solution of OCAP to the solution of a suitable path problem on a network whose size is kept under control. Another contribution is that of showing, through computational tests on real instances that our solution algorithm to OCAP scales pretty well to the case where we cast it into a dynamic sequential decision process where orders arrive over time. Last but not least, we make available to the scientific community some real computational instances as well as some insights and figures on meal delivery in Rome collected on the ground.

This work is organized as follows. In Section 2 we provide a literature review of some problems related to our application. Section 3 presents the specific problem encountered by $\mathcal{M}$, outlines the sequential decision process, and provides a formal definition for OCAP. In Section 4, we deal with our solution approach to solve OCAP. We first provide a suitable time-extended network, whose careful construction is discussed in Sections 4.1 and 4.2. We then show in Section 4.3 how OCAP can be reduced to the solution of a suitable path problem on this time-extended network and in Section 4.4 how the latter problem can be formulated as an Integer Linear Programming problem. We devote Section 5 to provide more details on the sequential decision process for the dynamic setting. Section 6 presents the results of our computational tests for both the offline and the dynamic scenario. Finally, Section 7 includes conclusions and directions for future research.

Disclaimer. In the paper, we often deal with numbers such as the number of orders, the numbers of couriers etc. These numbers are mainly from 2019 and are not at all representative of the current volumes handled by $\mathcal{M}$.

2. Literature

According to a well-recognized classification introduced in [2], Meal Delivery Problems (MDPS) fall in the wide area of the Transport On Demand (TD) problems. In particular, MDPS belong to a subclass of TD problems, that of Pickup and Delivery Problems (PDP), where a fleet of vehicles must transport individuals or goods from a set of pickup locations to certain delivery places. Even more, the MDP belongs to a subclass of PDPS, namely that of Vehicle Routing Problem with Pickup and Delivery (VRPPD). The peculiarity of VRPPD is that each transportation request has a single pickup place and a single delivery place.

According to such classification, MDPS are VRPPDS where food is transported. Note that a well-known subclass of the dynamic VRPPD problem is the dial-a-ride problem (DARP), where individuals are transported instead of goods. Unlike *standard* VRPPD, in both MDP and DARP it is often the case that not all client demands need to be satisfied. Moreover, while for standard VRPPD the objective function usually deals with total routing cost, both MDP and DARP deal with some measure of customer satisfaction.

As we pointed out before, the main distinction between MDP and DARP is that for the former problem goods (food) are transported and for the latter individuals. This is not just a *formal distinction*, as for MDP time windows are exceptionally narrow, making it hard to combine orders from different restaurants (as observed in [3]). According to [4], in MDPS, consecutive pickups for different customers are often not allowed, severely restricting the design of efficient routing plans; that is not often the case for DARPS (for a comprehensive overview of DARP see [5–8]).

Finally, we observe that the MDPS is also related to the so-called Same-day Delivery Problem (SDDP) [9,10], where consumers place orders to be delivered by a single vehicle or by a fleet of vans on the same day. In the SDDP there is usually a single hub storing all the required goods, all orders share a common deadline and, since the goods are not perishable, it is possible to consolidate a large number of requests to (almost) saturate the vehicle(s) capacity, reducing the number of empty trips to the depot. There is a wide literature related to the SDDP, see for instance [11–17].

If we now step back to TD problems, we observe that TD problems are usually classified as static if all requests are known in advance or dynamic if requests are received over time and routes must be adjusted dynamically. A more refined classification [18] partitions TD problems into several classes depending on the evolution and quality of information available: static-deterministic, static-stochastic, deterministic-dynamic, or dynamic-stochastic.

According to the above classification, the problem we are interested in here is a deterministic-dynamic MDP where part of the input is unknown and revealed over time. We, however, highlight that while our problem falls in the broad class of MDPS, several additional constraints are peculiar to our problem and will be discussed in Section 3. We, therefore, devote the last part of this section to reviewing a few papers in the specific literature on Meal Delivery Problems. One of the first contributions is given in [19] where the authors define a particular problem in the class of MDPS that they call the "Meal Delivery Routing Problem" (MDRP).

In the MDRP, orders from the same restaurant can be grouped into bundles that have to be assigned to couriers. Couriers have been given active time windows and are assumed to wait at the last drop-off location before moving to the next order to be delivered. Customers are not allowed to request a specific delivery time, each order has a maximum amount of time within which it must be delivered. Different

objective functions are considered, like minimizing the total cost for the courier (that can be paid according to the number of assigned orders or also considering a minimum amount for the work shift), or the average difference between delivery time and placement time of the orders. The authors introduce a fully deterministic dynamic model to address tasks such as dispatching orders to couriers, determining optimal routes, and sizing the fleet of drivers. In their pursuit of solving large-scale MDRP instances, they employ a heuristic solution algorithm based on a rolling-horizon repeated matching approach. Their study shows that it is possible to attain high-quality solutions across various service objectives, despite the simplicity and myopic nature of the proposed solution approach. The work in [20] develops an exact solution approach, based on a MILP model, for the deterministic meal-delivery routing problem (MDRP). To model the MDRP as a deterministic MILP, the authors assume perfect information about the order-arrival stream.

The "Virtual Food Court Delivery Problem" (VFCDP) proposed in [21] focuses on the assignment of couriers to sequences of pickups and deliveries with the objective of maximizing customer satisfaction while respecting committed delivery times. Also in the VFCDP, customers cannot request a delivery time but the company may commit to delivering the food later than the delivery time associated with the "as soon as possible" scenario. To solve this problem, the authors propose a MILP model and an auction-based heuristic. Similarly to [19], they evaluate the quality of the proposed approach simulating the dynamic setting characterizing the actual problem. In these simulations, the arrival of each new order triggers a re-optimization which provides an updated schedule for the operating couriers.

In more recent years, several other meal delivery models have been proposed. In [22] the authors investigate a problem in which orders are revealed over time but the company knows in advance the probability distribution of future requests and aims at minimizing the expected delays. Auad et al. [23] address a meal delivery problem where orders arrive dynamically over time, focusing on both customer and courier satisfaction. In their study, couriers handle deliveries within a region that can be dynamically adjusted during their working shifts. Xue et al. [24] propose a two-stage model where in the first stage the goal is to minimize the number of routes performed by couriers and in the second stage is to assign available riders to the planned routes. Liao et al. [25] study a multi-objective problem to maximize customer satisfaction and rider balance utilization, and to minimize carbon footprint. Bozanta et al. [26] proposes a reinforcement learning approach to solve small instances of a food delivery service problem, in which the goal is to maximize the company's revenues. These studies assume that couriers cannot reject assigned orders. In contrast, [27] explore the possibility that couriers may decline delivery requests. They analyze the impact of a platform that approximates and integrates individual couriers' acceptance behavior into the order dispatching process, quantifying its effects on all stakeholders, including the platform, customers, and couriers.

Moving to complexity issues [1,28] analyze some basic optimization problems that are relevant for MDPS and in particular for the model discussed in this paper (see Section 3.3). In the simplest model, a single restaurant entrusts a single courier to deliver meals to customers who place orders at the restaurant; Cosmi et al. [29,30],Agnetis et al. [31] elaborate on that model and propose exact solution approaches, based on integer linear programming and combinatorial branch and bound and prove that the single courier single restaurant problem is NP-HARD if orders may be delivered in bundles. Still, on the complexity side [32,33] discuss the hardness of a model where customers aim to receive their food as soon as possible and therefore a tardiness function has to be optimized.

To summarize the literature on meal delivery problems, Table 1 highlights its most relevant features and provides a classification of the mentioned works according to the classification for routing problems proposed in [18]. We point out that in this paper we provide both static and dynamic deterministic models. Our solution algorithms, see the following, are based on Integer Linear Programming and, as we later show, scale pretty well. Moreover, our models allow the customers to define time windows for the delivery and take into account quality of service issues. Ulmer et al. [22] propose a similar work. However, they assume to have knowledge of the probability distribution of meal preparation times and order arrival times and locations. On the contrary, we assume that there is no knowledge of future orders' arrival probability distribution and meal preparation times are deterministic and provided by the restaurant when the order is received. These two assumptions come from our collaboration with $\mathcal{M}$ and their daily operational experience.

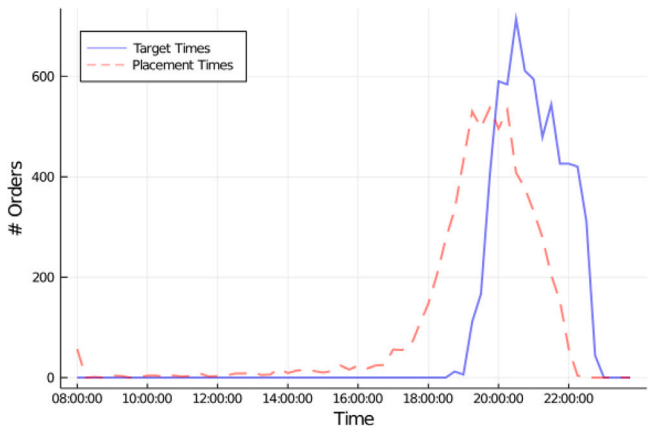

Fig. 1. For each time instant, the overall number of orders that in 2019 were placed at that time instant (red, dashed) and the overall number of orders requiring that time instant as target times (blue, solid) for dinner services. Customers may ask only for a delivery time between 19:00 and 22:30 and can place orders in advance. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3. Problem setting

3.1. Towards an optimization model for courier assignment

We start providing a few data and figures concerning the customer demand collected by $\mathcal{M}$ from January to March 2019. We focus on the dinner service, as $\mathcal{M}$ receives a larger number of orders in that service. We aim to motivate some assumptions at the base of the optimization models presented later.

We start with Fig. 1, where we report for each time instant, the overall number of orders that in 2019 were placed at that time instant (red) and the overall number of orders requiring that time instant as target times (blue) for dinner services. This figure hence provides the distribution of *placement times*, i.e., the time at which a customer places an order, vs *target times*, i.e., the time at which that customer would like to receive that order. Note that, even though Fig. 1 deals with orders for the dinner service, customers might place orders for that service well in advance, e.g. in the morning (actually even days in advance, but that does not happen in reality). Fig. 2 and Table 2 build upon Fig. 1. Fig. 2 plots the difference between the placement time and the target time of the same order. Note that the policy of $\mathcal{M}$ is such that the minimum slack between the placement time and its target time is 30 min; however, a large fraction of customers prefer to order in advance. Table 2 summarizes these data by showing the percentage and the cumulative percentage of orders whose target time is within n minutes from their placement times, for different values of n. *We point out that about 70% of the orders are placed more than 1 h in advance.*

We are now ready to shape an optimization model for the courier assignment. As we already discussed in the introduction, $\mathcal{M}$ focuses on top-end customers and pursues high Quality of Service: in particular

Table 1
Meal delivery literature: columns "Restaurants" and "Couriers" report if the study considers a single (S) or multiple (M) vehicles or restaurants, "TW" reports if time windows chosen by customers are considered (✓means yes, ✗means no), "QoS" reports if the considered objective tries to maximize the QoS (e.g, max number of on-time deliveries, min (weighted) function of delays), "Scalability" reports if the proposed solution can solve large real-world problem.

Classification	Authors (year)	Restaurants	Couriers	TW	QoS	Scalability
Static-Deterministic	Yildiz and Savelsbergh (2019) [20]	M	M	✗	✓	✗
	Cosmi, Oriolo et al. (2019) [1]	S-M	S-M	✓	✗	✗
	Cosmi, Nicosia and Pacifici (2019) [29]	S	S	✓	✗	✗
	Cosmi, Nicosia and Pacifici (2019) [30]	S	S	✓	✗	✗
	Liao, Zhang and Wei (2020) [25]	M	M	✓	✓	✗
	Xue, Z. Wang and G. Wang (2021) [24]	M	M	✓	✗	✗
	Joshi et al. (2021) [32]	M	M	✗	✓	✗
	Böhm, Megow and Schlöter (2022) [34]	S-M	S-M	✓	✗	✗
	This work	*M*	*M*	✓	✓	✓
Dynamic-Stochastic	Steever, Karwan and Murray (2019) [21]	M	M	✗	✓	✓
	Ulmer, Thomas, Campbell et al. (2020) [22]	M	M	✗	✓	✓
	Bozanta et al. (2022) [26]	M	M	✗	✗	✗
Dynamic-Deterministic	Reyes et al. (2018) [19]	M	M	✗	✓	✓
	Agnetis et al. (2023) [31]	S	S	✓	✗	✗
	Auad, Erera and Savelsbergh (2024) [23]	M	M	✗	✓	✓
	This work	*M*	*M*	✓	✓	✓

Table 2
Percentage and cumulative percentage of orders with a target time within n minutes from the placement time (the slack between target time and placement time of a same order cannot be less than 30 min).

	[30, 45)	[45, 60)	[60, 75)	[75, 90)	[90, 105)	[105, 120)	> 120
%	11.2	20.5	26.8	14.4	5.3	3.8	18.0
Cumulative%	11.2	31.7	58.5	72.9	78.2	82.0	100.0

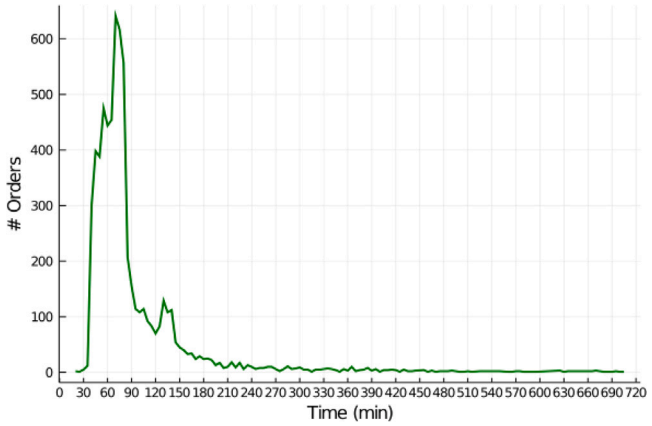

Fig. 2. On the x-axis the distance between the placement time and the target time of a same order, on the y-axis the number of orders placed during that service.

through positive commitment of couriers and careful management of delays. Fundamental facts about couriers are the following: (i) couriers are hired before the service, and only a small part of their salary is paid on a piecework basis; (ii) a courier that has been assigned a delivery cannot refuse it; (iii) a courier may decide when to start her work shift, but must be available until the end of the service; (iv) a courier will autonomously choose her route for dispatching an order that has been assigned. Note also that, since couriers cannot carry multiple orders at the same time, the typical route of a courier will follow a sequence such as pickup of order i_1 - delivery of order i_1 - pickup of order i_2 - delivery of order i_2 ....

As for the management of delays, the key fact is that customers who experience a delay, or even worse a rejection, will be given a discount voucher for future orders: the value of the voucher is proportional to the delay they suffered and higher in case of rejections (recall that orders with delay larger than 60 min will always be rejected, see Section 1). Therefore, we aim at dispatching orders to couriers to avoid as many as possible delays and rejections.

A last key fact follows from the nature of the demand. As discussed above, see Table 2, only a relatively small number of orders are placed on a best-effort basis (that is, customers asking for a delivery as soon as possible). Therefore, we do not try to anticipate demand or make assumptions about the distribution of orders.

In agreement with the management of $\mathcal{M}$ we then decided to develop a sequential decision process handled using a rolling horizon approach building upon the solution of a *fully deterministic* optimization problem, that is called the *Offline Couriers Assignment Problem* (OCAP). In defining OCAP we indeed assume full knowledge of the set of orders to be delivered; then we are interested in dispatching these orders to couriers to minimize the cost of delays and rejections. The approach then casts OCAP into a sequential decision process depicted in Fig. 3.

Following Soeffker et al. [35], we represent our sequential decision process by a sequence of states. Each state s_k corresponds to an instance I^k of OCAP that is built on the base of the previous state (instance), and based on the orders placed during a suitable time window connected to s_k. The solution of the instance I^k of OCAP – e.g. through Integer Linear Programming – provides decisions that contribute to the definition of the next instance I^{k+1}. Therefore, we build and solve a sequence of instances $I^0, I^1, \dots$ of OCAP, such that I^{k+1} inherits from the solution to I^k data and decisions, while adding the set of orders that have been placed in the last time window. We will provide more info about the sequential decision process in Section 5, while in the next sections, we focus on the solution of OCAP.

3.2. The offline couriers assignment problem

We start by providing some basic definitions and discuss a few more assumptions that are at the base of OCAP. In the following, for T a positive integer, we let $[T]$ denote the set $\{0, 1, 2, \dots, T\}$.

Discretization. We discretize time and each service, i.e., lunch or dinner, can be thought of as a finite set of *time steps* $\{0, 1, 2, \dots, T\}$ each corresponding to a *time instant* $t^h = t^0 + h \cdot \gamma$, for $h \in [T]$, where $\gamma \in \mathbb{Z}_+$ is the *base step*. Time step 0 corresponds to the time instant t^0 that designates the start of the service (in our case, 18:30), while time step T corresponds to the last time instant that can be chosen by customers for delivery (in our case, 22:30). *In the following, we therefore refer to*

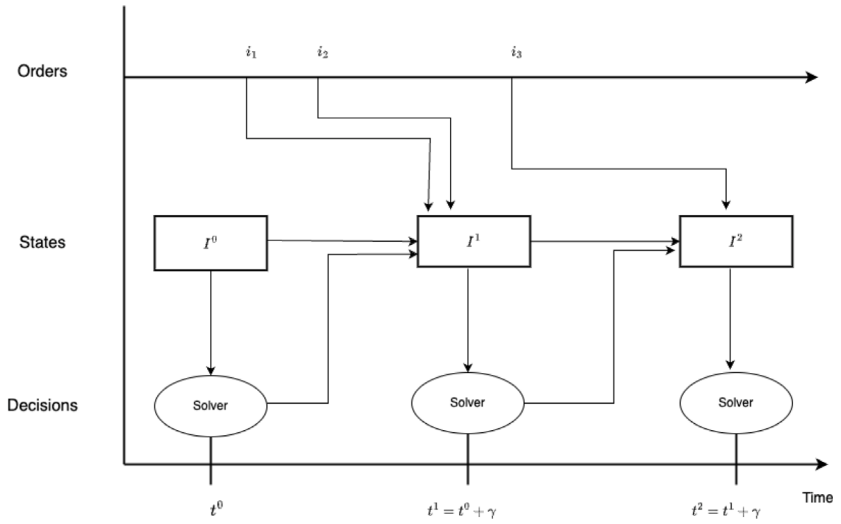

Fig. 3. We graphically represent our sequential decision process. The states correspond to different instances of OCAP, which are influenced by the orders placed along the time line and by the previous states, whereas the decisions are the result of applying the solver on the current instance.

time steps rather than time instants. E.g. the target time $t' \in T$ at which a customer would like to receive the delivery (see below) is indeed the closest time step to the "true" time instant θ' required by the customer. Analogously, the average time that a courier has to wait at a restaurant to collect an order (see below) is also expressed in the number of time steps.

Orders. We are given a set of orders O. Each order $i \in O$ has been placed by a customer and, as discussed in Section 1, the orders cannot be aggregated with other orders. Therefore, for each order $i \in O$, we are given:

a *pickup location* p_i, which is where couriers will collect the order (usually, the address of a restaurant);

a *delivery location* d_i (usually, the address of the customer);

a *target* time $t'_i \in [T]$ that is the time at which the customer would like to receive the delivery; note however that the smaller values in $[T]$ cannot be chosen as couriers use those times for setting up for the service and e.g. reaching the location of a pickup;

the *average waiting time* $\beta_i \in [T]$, the average time that a courier has to wait once arrived at p_i to have the order ready (As a matter of fact, quite often couriers do indeed *wait* at a restaurant to collect an order, even though restaurants are always informed of – and agree on – the pickup time of an order: that is because restaurants want to be sure that an order will not be ready *before* the courier shows up, as to avoid that the quality of food to degrade).

Note that, for each order, there is indeed also a *placement time*, as it was defined in Section 3.1, however, this value is not interesting for the solution of an instance of OCAP.

Width. While for each order i we are given a target time t'_i for its delivery, we, however, allow for the possibility of delivering at a time $t_i \in [T]$ different from t'_i, provided that t_i belongs to a suitable time window around t'_i. We are therefore given two parameters, $w'_l, w'_r \in \mathbb{Z}_+$, so that the order can be indeed delivered at any time in $\{t'_i - w'_l, t'_i - w'_l + 1, \dots, t'_i + w'_r - 1, t'_i + w'_r\}$. Note that we allow early deliveries. However, the time window needs not to be symmetric with respect to t' and very likely $w'_l \ll w'_r$. We let $w := w'_l + w'_r$ be the *width*. In practice, the width is a parameter of capital importance: the smaller the value of w is, the higher the QoS is since the delivery of i will be more likely at a time close to t'_i.

It follows that we may assume that for each order $i \in O$, we are also given:

a *time window* $W_i = \{t_i, t_i + 1, \dots, t_i + w\} \subset [T]$, where $t_i := t'_i - w'_l$ is the *start time* and w is the width just defined. So order i can be delivered at any time in W_i (with different costs, though: see the following).

Rejection cost. As we discussed in Section 3.1, we aim at minimizing a suitable cost function that takes into account delays and rejections, without considering any cost related to the number of couriers (as they are hired before the service) and to the cost of the routing (as the couriers will autonomously choose their routes). An order $i \in O$ that cannot be delivered at a time $t \in W_i$ will be rejected with a rejection cost $\bar{c}$.

Delay cost function. We may incur some costs even if we deliver an order i at a time $t \in W_i$. There are indeed different costs for delivering i at different times of W_i. Namely, we are given:

a non-decreasing *delay cost-function* $c : [w] \mapsto \mathbb{Z}_+$, such that, for $h \in [w]$, the cost of delivering i at time $t_i + h$ is equal to $c(h)$.

Assuming that the delay cost function c is non-decreasing, i.e., is such that $c(h) \leq c(h+1)$ for each $h \in [w-1]$, is quite natural. However, recall that we allow for early deliveries, i.e., deliveries can be made before the target time requested by the customer. In this case, the hypothesis makes sense as long as we do not deliver too far in advance.

Couriers. We are given a set C of couriers hired (in advance) for the service. For each $j \in C$, we are given:

a *release location* p_j;

a *release time* $r_j \in [T]$.

The courier j will start her work shift from location p_j at time r_j. Note that it is possibly the case that $r_j > 0$, i.e. the courier does not start her work shift at the beginning of the service but later. To the contrary, we assume that she will be available for deliveries up to time T and without any constraint on the location of her last delivery: possibly at a location far from p_j. As we will later discuss, our flow-like integer linear programming formulation (see Section 4.4) builds upon the fact that somehow couriers cannot be distinguished from each other once they start their work shift.

Travel Times and **Locations.** It follows from above that we can infer a set L of relevant *locations*, namely $L := \{p_i, i \in O\} \cup \{d_i, i \in$

$O\} \cup \{p_j, j \in C\}$. We therefore assume that, for each ordered pair of locations (x, y), $x, y \in L$, we are given:

the *expected travel time* $d(x, y) \in [T]$ to go from x to y.

As discussed in Section 3.1, a courier that has been assigned an order $i \in O$ will autonomously choose her route for going from pickup location p_i to delivery location d_i. However, the expected travel time $d(p_i, d_i)$ is the same for each courier. This assumption is extremely reasonable in our case study since all the couriers in Rome travel by motorbikes whose travel times are not affected by traffic conditions.

Check Out Time. Finally, we assume that we are given:

$\alpha \in [T]$, the *check-out time*, i.e. the average time for a courier to check out and be ready for the next delivery (if any).

3.3. A formal definition of OCAP

We are now ready to provide a formal definition of OCAP. Following the discussion in the previous section, an instance of the problem is defined by a tuple $(O, C, L, w, c, \bar{c}, \alpha, d)$. We want to solve the problem of delivering the orders in O at a minimum cost in an *offline fashion*, i.e., assuming full knowledge of O. In particular, we ignore the placement time of the order and assume that all the orders are known at the beginning of the shift. There are two different sub-problems that need to be concurrently addressed: the assignment of orders in O to couriers, i.e., an assignment of subsets $O_j \subseteq O$ to each courier $j \in C$ such that the sets O_j are pairwise disjoint; the scheduling of the orders in O_j, for each courier $j \in C$. In solving these problems, we must take into account several constraints, e.g. related to time windows, and minimize the overall cost of the delivery, which takes into account both the delay cost function c for orders that are delivered and the fixed rejection cost $\bar{c}$ for orders that are rejected.

In order to provide a formal statement of the problem, we need a few definitions. First, consider a pair of orders $i, i' \in O$ and a courier $j \in C$.

- We denote by $\theta(i, j)$ the minimum time at which j could deliver i if the latter is the first order delivered by j in her shift. Therefore: $\theta(i, j) := r_j + d(p_j, p_i) + \beta_i + d(p_i, d_i)$.
- We denote by $\theta(i, i', y)$ the minimum time at which order i can be delivered assuming that: (1) it is delivered right after order i' and by the same courier; (2) order i' is delivered at time y. Therefore: $\theta(i, i', y) := y + \alpha + d(d_{i'}, p_i) + \beta_i + d(p_i, d_i)$

A *schedule* for a courier $j \in C$ is a pair (O_j, σ_j), where O_j is a subset of O and $\sigma_j : O_j \mapsto \{r_j, r_j + 1, \dots, T\}$ is a function that associates with each order $i \in O_j$ the time $\sigma_j(i)$ at which courier j delivers order i. We assume without loss of generality that $O_j = \{i^j_1, \dots, i^j_{|O_j|}\}$, with $\sigma_j(i^j_{k-1}) \leq \sigma_j(i^j_k)$, for $k = 2..|O_j|$. Following the above discussion, it follows that (O_j, σ_j) is *feasible* if:

(i) $\sigma_j(i^j_k) \in \{t_{i^j_k}, t_{i^j_k} + 1, \dots, t_{i^j_k} + w\}$, $k = 1, \dots, |O_j|$;
(ii) $\sigma_j(i^j_1) \geq \theta(i^j_1, j)$;
(iii) $\sigma_j(i^j_k) \geq \theta(i^j_k, i^j_{k-1}, \sigma_j(i^j_{k-1}))$.

Constraint (i) requires that $\sigma_j(i^j_k)$ is within the time window $W_{i^j_k}$ of order i^j_k (recall that we denote by $t_{i^j_k}$ the delivery start time for order i^j_k). Constraint (ii) requires that $\sigma_j(i^j_1)$, the delivery time for the first order i^j_1 assigned to j, is consistent with the release time of j. Constraint (iii) analogously requires that $\sigma_j(i^j_{k+1})$, the delivery time for order i^j_{k+1}, is consistent with the delivery time of i^j_k, which is the previous order in the schedule of j. We illustrate the definition of schedule on a simple example. We deal with the case of a courier O that has been assigned three orders.

Example 1. We are given a courier 0 with release time $r_0 = 0$ and a set O of three orders $\{1, 2, 3\}$, respectively with target times $t_1 = 6$, $t_2 = 12$ and $t_3 = 15$ and average waiting times $\beta_1 = \beta_2 = \beta_3 = 1$ (see Fig. 4). The width w is equal to 15, the check-out time α is equal to 1 and expected travel times are: $d(p_0, p_1) + d(p_1, d_1) = d(p_0, p_2) + d(p_2, d_2) = d(p_0, p_3) + d(p_3, d_3) = 5; d(d_1, p_2) + d(p_2, d_2) = 4; d(d_2, p_1) + d(p_1, d_1) = 5; d(d_1, p_3) + d(p_3, d_3) = d(d_3, p_1) + d(p_1, d_1) = 7; d(d_2, p_3) + d(p_3, d_3) = d(d_3, p_2) + d(p_2, d_2) = 2$.

For the case of simplicity, we restrict to schedules where the courier will anyhow deliver order 1 first: this will be done not earlier than time 6, as $r_0 = 0$, $d(p_0, p_1) + d(p_1, d_1) = 5$ and $\beta_1 = 1$. There are then two possible cases: (*i*) the courier delivers order 2 before order 3; (*ii*) the courier delivers order 3 before order 2.

(*i*) In this case, the courier will deliver order 2 not earlier than time 12 because $6 + \alpha + d(d_1, p_2) + d(p_2, d_2) + \beta_2 = 12$. She will finally deliver order 3 not earlier than time 16 because $12 + \alpha + d(d_2, p_3) + d(p_3, d_3) + \beta_3 = 16$. Note that, in this case, it is therefore possible to deliver the first two orders at their target time and the third order just one time step later than its target time.

(*ii*) In this case, the courier will deliver order 3 not earlier than time 15 because $6 + \alpha + d(d_1, p_3) + d(p_3, d_3) + \beta_3 = 15$. She will finally deliver order 2 not earlier than time 19 because $15 + \alpha + d(d_3, p_2) + d(p_2, d_2) + \beta_2 = 19$. Note that, in this case, it is therefore possible to deliver the first and the third order at their target time but the second order cannot be delivered earlier than 7 time steps after its target time.

We now move to costs. Note that the cost of delivering i^j_k at time $\sigma_j(i^j_k)$ is $c(\sigma_j(i^j_k) - t_{i^j_k})$. We therefore let $c(O_j, \sigma_j) := \sum_{k=1,\dots,|O_j|} c(\sigma_j(i^j_k) - t_{i^j_k})$. We aim at finding a collection $(\mathcal{O}, \sigma) = \{(O_1, \sigma_1), \dots, (O_{|C|}, \sigma_{|C|})\}$ such that each $\sigma_j : O_j \mapsto \{r_j, r_j + 1, \dots, T\}$ is a feasible schedule for courier $j \in C$, the sets O_j are pairwise disjoint and the cost $c(\mathcal{O}, \sigma)$ is minimized. The latter cost is the sum of two terms: the cost $\sum_{j \in C} c(O_j, \sigma_j)$, that is the cost of orders that are indeed delivered, and the cost $\bar{c} \cdot |O \setminus \bigcup_{j \in C} O_j|$, that is the cost of orders that are rejected. Namely, the *Offline Couriers Assignment Problem* is defined as follows:

The Offline Couriers Assignment Problem (OCAP). Given: a tuple $(O, C, L, w, c, \bar{c}, \alpha, d)$, as defined in Section 3.3. **Find:** a collection of schedules $(\mathcal{O}, \sigma) = \{(O_1, \sigma_1), \dots, (O_{|C|}, \sigma_{|C|})\}$ such that: each $\sigma_j : O_j \mapsto \{r_j, r_j + 1, \dots, T\}$ is a feasible schedule for courier $j \in C$; the sets O_j are pairwise disjoint; the cost $c(\mathcal{O}, \sigma) := \bar{c} \cdot |O \setminus \bigcup_{j \in C} O_j| + \sum_{j \in C} c(O_j, \sigma_j)$ is minimized.

We point out that when the width is unbounded (the decision version of) OCAP is NP-HARD already for the case of a single courier, through the reduction of a classical scheduling problem: Sequencing with Release Times and Deadlines (see [1,36]).

4. A solution approach to OCAP

In this section, we show how to reduce OCAP to the problem of finding a collection $\mathcal{P}$ of suitable paths in a time-extended network $\mathcal{Q}(N, A)$. We will show in Section 4.4 that the latter problem can be formulated as an Integer Linear Program whose size strictly depends on the number of arcs in A. In order to keep the latter number under control, we focus on a particular class of schedules, that, in accordance with the scheduling literature, we call *early start*.

4.1. Early start schedules

Let $O_j = \{i^j_1, \dots, i^j_{|O_j|}\}$ be the set of orders assigned to a courier $j \in C$ and let $\sigma : O_j \mapsto \{r_j, r_j + 1, \dots, T\}$ be such that (O_j, σ_j) is a feasible schedule. We assume without loss of generality that $O_j = \{i^j_1, \dots, i^j_{|O_j|}\}$, with $\sigma_j(i^j_{k-1}) \leq \sigma_j(i^j_k)$, for $k = 2..|O_j|$. The schedule (O_j, σ_j) is an *early start* schedule if σ_j is such that the courier j delivers each order as soon as possible, i.e., without unnecessary idle times with respect to the sequence $i^j_1, \dots, i^j_{|O_j|}$. Namely, (O_j, σ_j) is a (feasible) early start schedule if the followings hold:

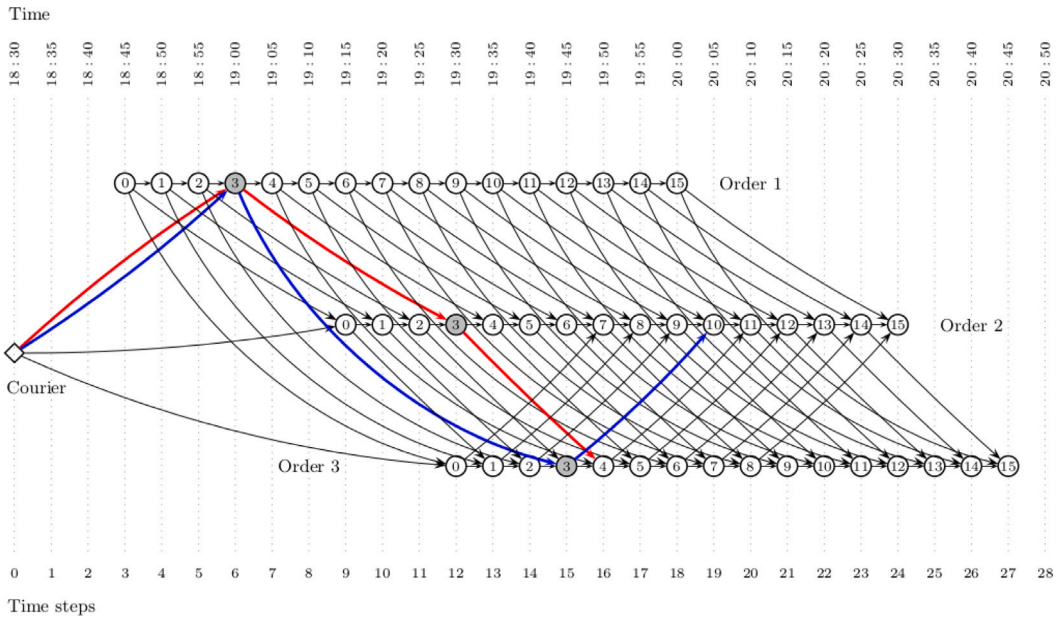

Fig. 4. The time extended network associated with the instance described by Example 1. The three orders $\{1, 2, 3\}$ are associated with the layers with the circled nodes. Each of these nodes is labeled with h, for $h \in [15]$, and the nodes corresponding to target times are in gray. The courier is represented by the diamond node placed at time step 0 (i.e., real time 18:30). We restrict to schedules where the courier will anyhow deliver order 1 first; therefore the arcs from the second and the third layer to the first one are not useful and we do not draw them for the sake of clarity. The red path represents the early start schedule where the courier serves the orders according to the sequence (1, 2, 3) (case (i) in Example 1). The blue path represents instead the early start schedule where the courier serves the orders according to the sequence (1, 3, 2) (case (ii) in Example 1). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

(i) $\sigma_j(i^j_k) \in \{t_{i^j_k}, t_{i^j_k} + 1, \dots, t_{i^j_k} + w\}$, for each $k = 1, \dots, |O_j|$;

(ii) $\sigma_j(i^j_1) = \max\{t_{i^j_1}, \theta(i^j_1, j)\}$;

(iii) $\sigma_j(i^j_{k+1}) = \max\{t_{i^j_{k+1}}, \theta(i^j_{k+1}, i^j_k, \sigma_j(i^j_k))\}$, for each $k = 1, \dots, |O_j| - 1$.

If we go back to Example 1 we observe that the delivery sequence $(1, 2, 3)$ can be implemented by different schedules for the courier (note that in the following we drop the subscript for σ as there is only one courier): we may e.g. set $\sigma(1) = 6, \sigma(2) = 13$ and $\sigma(13) = 20$. However, for that sequence, there is only one early start schedule, which is: $\sigma(1) = 6, \sigma(2) = 12$ and $\sigma(13) = 19$.

We skip the straightforward proof of the following lemma. Similar results are well-known in the scheduling literature.

Lemma 2. *Since the delay cost function c is non-decreasing, it follows that there exists an optimal solution to* OCAP *where, for each courier, the schedule is early start.*

Remark 3. We point out that even for an optimal early start solution, i.e., a collection of optimal schedules $(\mathcal{O}, \sigma) = \{(O_1, \sigma_1), \dots, (O_{|C|}, \sigma_{|C|})\}$ such that each pair (O_j, σ_j) is a feasible early start schedule, it is possible that for a courier j there are some unavoidable *idle times*. For instance, this happens if $t_{i^j_1} > \theta(i^j_1, j)$, as in this case, the courier j will have an idle time of size $t_{i^j_1} - \theta(i^j_1, j)$ before her first delivery. In practice, she has to be at the pickup location of i^j_1 by time $t_{i^j_1}$ and may spend that idle times wherever she likes: at her release location, or at the pickup location of i^j_1, fractionally at both locations or in any point in between the two locations. The same holds if $t_{i^j_{k+1}} > \theta(i^j_{k+1}, i^j_k, \sigma_j(i^j_k))$, for some $k = 1, \dots, |O_j| - 1$.

4.2. A time extended network

We show in the following that we may reduce OCAP to the problem of finding a suitable collection $\mathcal{P}$ of paths in a *time extended* network $Q(N, A)$. This network will have a horizontal *layer* with $w + 1$ nodes for each order i (recall that w is the width): namely, each layer is a directed path with $w + 1$ nodes, each corresponding to the delivery of order i at a time step in its time window. Different horizontal layers will be connected through some *diagonal* arcs: an arc going from layer i to layer i' represents the fact that order i' can be delivered right after order i by some courier. It will be indeed possible to associate with each courier j some paths of Q: each path starts from a single node representing j and visits a sequence of layers representing the sequence of orders assigned to, and delivered by j. We finally point out that, since we may restrict to an early start schedule (see Lemma 2), we can keep the number of diagonal arcs between each pair of layers of size $O(w)$: namely, each node u in the layer of i may be connected to at most *one* node v of layer i': namely v corresponds to deliver i' as soon as possible, i.e., without unnecessary idle times, with respect to delivering i at the time step defined by u. Note that therefore the set of arcs going from a layer i to a layer i' will be made of "parallel" arcs, as when there are no idle times each shift in the delivery time for i will induce the same shift in the delivery time for i'.

We are given an instance $(O, C, L, w, c, \bar{c}, \alpha, d)$ of OCAP as defined in Section 3.3. We first show how to build the time extended network $Q(N, A)$ associated with this instance. We start with defining the set of nodes N. We associate with each order $i \in O$ a set of $w+1$ nodes that we call the *layer* of i. Namely, the layer of i is made of nodes $(i, 0), \dots, (i, w)$, and the node (i, h), for $h \in [w]$, stands for the delivery of order i at time $t_i + h$.

We also associate with each courier $j \in C$ a node that is convenient to denote as $(j, 0)$. The node $(j, 0)$ stands for j being ready at her release location at time r_j. It is also convenient to think of $(j, 0)$ as the layer of courier j, even though in this case the layer is made of a single node.

We now define the set of arcs A. There are three sets of arcs, each with a cost, that will be discussed later. A first set of arcs, that we call *horizontal*, run in the layer of orders and stand for a courier waiting at the delivery location of an order: these arcs model unavoidable idle times, see Remark 3, and allow to keep the size of A under control by

reducing the number of *order diagonal* arcs, see below. Namely, for each order $i \in O$, we define the following set of arcs:

for $h \in [w-1]$, the arc $((i,h),(i,h+1))$, which stands for a courier waiting from time h to time $h+1$ at the delivery location of order i.

A second set of arcs, that we call *courier diagonal*, run from the layer of a courier to the layer of an order and stand for that order being the first delivered by that courier (in her shift). Namely, let $j \in C$ be a courier and $i \in O$ an order. Again, we denote by $\theta(i,j)$ the minimum time at which courier j could deliver her first order i, i.e. $\theta(i,j) := r_j + d(p_j,p_i) + d(p_i,d_i) + \beta_i$. There are three possible cases:

$\theta(i,j) > t_i + w$ In this case, courier j cannot deliver order i by time $t_i + w$, which is the last time for delivering it: there will be no arcs running from the layer of j to the layer of i;

$t_i + w \geq \theta(i,j) \geq t_i$ In this case, by leaving p_j at time r_j, courier j will reach d_i within the time window $[t_i, t_i+w]$: there will be a single arc from layer j to layer i and, according to the early start policy, it will be the arc $((j,0),(i,\theta(i,j)-t_i))$;

$t_i > \theta(i,j)$ In this case, by leaving p_j at time r_j, courier j will reach d_i before time t_i. There is therefore an *idle time* (see Remark 3) of size $t_i - \theta(i,j)$: there will be a single arc from layer j to layer i and, according to the early start policy, it will be the arc $((j,0),(i,0))$.

The last set of arcs, which we call *order diagonal*, runs from the layer of an order to the layer of another order and stands for an order being delivered right after the other by the same courier. Consider therefore orders i and l in O. Let $\theta(l,i,t_i) := t_i + \alpha + d(d_i,p_l) + \beta_l + d(p_l,d_l)$ be the earliest time at which a courier can deliver order l *if she delivers order* i *at time* t_i (which is the first possible delivery time for i). We delve into four cases:

$\theta(l,i,t_i) > t_l + w$ In this case, the courier cannot deliver order l by time $t_l + w$, which is the last time for delivering it: there will be no arcs running from the layer of i to the layer of l;

$t_l + w \geq \theta(l,i,t_i) \geq t_l$ In this case, if the courier delivers i at time t_i, she will reach d_l within the time window $[t_l, t_l + w]$. According to the early start policy, the arcs that are going from layer i to layer l are the following: $((i,0),(l,\theta(l,i,t_i)-t_l)),((i,1),(l,\theta(l,i,t_i)-t_l+1)),\dots,((i,w-\theta(l,i,t_i)+t_l),(l,w))$. Analogously, if she delivers i at a time $t_i + t_l - \theta(l,i,t_i) + \mu$, for any $\mu \in [w+\theta(l,i,t_i)-t_l]$, she will reach d_l at time $t_l + \mu$. Therefore, the complete set of arcs going from layer i to layer l is the following: $((i,t_l - \theta(l,i,t_i)),(l,0)),((i,t_l-\theta(l,i,t_i)+1),(l,1)),\dots,((i,w),(l,w+\theta(l,i,t_i)-t_l))$.

$t_l > \theta(l,i,t_i) \geq t_l - w$ In this case, if the courier delivers i at any time in $[t_i, t_i+t_l-\theta(l,i,t_i)]$ she will still be able to reach d_l by time t_l. There is therefore an idle time (see Remark 3) of size $t_l-\theta(l,i,t_i)$: this will be represented by the *single* arc $((i,t_l-\theta(l,i,t_i)),(l,0))$. Analogously, if the courier delivers order i at a time $t_i + t_l - \theta(l,i,t_i) + \epsilon$, for any $\epsilon \in [w+\theta(l,i,t_i)-t_l]$, she will reach d_l at time $t_l + \epsilon$. Therefore, according to the early start policy, the complete set of arcs going from layer i to layer l is the following: $((i,t_l-\theta(l,i,t_i)),(l,0)),((i,t_l-\theta(l,i,t_i)+1),(l,1)),\dots,((i,w),(l,w+\theta(l,i,t_i)-t_l))$.

$t_l - w > \theta(l,i,t_i)$ In this case, even if the courier delivers i at time t_i+w, which is the last time to deliver, she will reach d_l before time t_l. This case is indeed similar to the previous case and there is again an idle time and we again put a *single* arc from layer i to layer l, namely the arc $((i,w),(l,0))$.

We now deal with arc costs. Each arc is entering a node (i,h), for suitable $i \in O$ and $h \in [w]$. We give each arc a entering node (i,h) the cost $c_a := c(h)$ of delivering order i at time $t_i + h$, but for the arc $((i,h-1),(i,h))$ that has cost zero.

The time extended network for the instance described by Example 1 is shown in Fig. 4. If we again restrict to schedules where the courier will anyhow deliver order 1 first, diagonal arcs going from the second and the third layer to the first one are not useful so we do not draw them for the sake of clarity. Still, for the sake of clarity, we skip the cost of the arcs. We will later explain what the red and the blue paths represent.

We close this section by first discussing the size of the network $Q(N,A)$. The number of nodes is roughly $w\cdot|O|+|C|$. As for the number of arcs, we start with horizontal arcs: their number is roughly $w\cdot|O|\cdot|C|$. The number of courier diagonal arcs is at most $|O|\cdot|C|$, as there is at most one arc for each courier to the layer of each order. Finally, the number of diagonal arcs is at most $w\cdot|O|^2$: note that horizontal arcs allow to keep the number of diagonal arcs under control and to avoid $O(w^2)$ factors. It turns out that the size of w has a strong impact on the size of Q: that is not surprising since as we already pointed out, when the width is unbounded, (the decision version of) OCAP is NP-HARD already for the case of a single courier (see [1]).

4.3. OCAP *as a path problem on the time extended network*

Lemma 2 shows that there exists an optimal solution to OCAP where, for each courier, the schedule is early start. We now show that *each* feasible early start schedule of a courier corresponds to a suitable path in the network $Q(N,A)$ *and vice versa*.

We start with a couple of useful notations. Each node in Q is indexed by a suitable pair (i,h): we denote by $\delta^-(i,h)$ ($\delta^+(i,h)$) the set of arcs entering (leaving) the node (i,h). We then denote, for each order $i \in O$, by $\delta^-(i)$ the set of arcs "entering" the layer of nodes associated with i:

$$\delta^-(i) = \left(\bigcup_{h\in[w]} \delta^-(i,h)\right) \setminus \{((i,h),(i,h+1)), h \in [w-1]\},$$

i.e., $\delta^-(i)$ is the set of arcs entering the cut induced by the set of nodes $\bigcup_{h\in[w]}(i,h)$. By definition, the arcs in $\delta^-(i)$ are either courier diagonal or order diagonal.

Let $j \in C$ be a courier and (O_j,σ_j) a feasible early start schedule, with $O_j = \{i^j_1,\dots,i^j_s\} \subseteq O$ and $\sigma_j : O_j \mapsto [r_j, T]$. Recall that $t_{i^j_k}$ is the start time of order i^j_k, for $k = 1,\dots,s$. We want to associate with (O_j,σ_j) a path P_j through the nodes:

$$(j,0),(i^j_1,\sigma_j(i^j_1)-t_{i^j_1}),(i^j_2,\sigma_j(i^j_2)-t_{i^j_2}),\dots,(i^j_{s-1},\sigma_j(i^j_{s-1})-t_{i^j_{s-1}}),(i^j_s,\sigma_j(i^j_s)-t_{i^j_s})$$

but we must take into account that it is possibly the case that there is no arc between two consecutive nodes in the above sequence. On the one hand, that happens only if the schedule (O_j,σ_j) has some idle times (see Remark 3) and, in this case, there is a path between the two nodes that exploits horizontal arcs. We therefore associate with (O_j,σ_j) the path P_j:

$$(j,0),(i^j_1,\sigma_j(i^j_1)-t_{i^j_1}),\dots,(i^j_1,\sigma_j(i^j_1)-t_{i^j_1}+b_1),(i^j_2,\sigma_j(i^j_2)-t_{i^j_2}),$$
$$\dots,(i^j_2,\sigma_j(i^j_2)-t_{i^j_2}+b_2),\dots$$
$$\dots,(i^j_{s-1},\sigma_j(i^j_{s-1})-t_{i^j_{s-1}}),\dots,(i^j_{s-1},\sigma_j(i^j_{s-1})-t_{i^j_{s-1}}+b_{s-1}),(i^j_s,\sigma_j(i^j_s)-t_{i^j_s}).$$

for some suitable $b_k \in [w-\sigma_j(i^j_k)]$ and for each $k = 1,\dots,s-1$: for each $b_k > 0$, the path P_j exploits horizontal arcs in the layer of order i^j_k to represent idle times.

Note that, while the path P_j might visit several nodes from the layer of the same order, these nodes will be visited one after the other, and the layer of each order is indeed visited by P_j at most once. In other words, the following holds:

Remark 4. For each order $i_k^j \in O_j$, there is exactly one arc in $P_j \cap \delta^-(i_k^j)$: the arc is courier diagonal if i_k^j is the first order delivered by j, and order diagonal otherwise.

Vice versa, consider now a path P_j that is going from node $(j,0)$ to node (i,h), for some $j \in C, i \in O$ and $h \in [w]$ and obeys to Remark 4. We also assume without loss of generality that either $h = 0$ or $(i, h-1) \notin P_j$ (otherwise we just shorten the path). Then the sequence of nodes visited by P'_j must be of the following form:

$\{(j,0),(i_1,h_1),\dots,(i_1,h_1+b_1),(i_2,h_2)\dots,(i_{s-1},h_{s-1}+b_{s-1}),(i_s,h_s)\}$ with:

- $s \geq 1, i_s = i$, $i_k \in O$ and $h_k \in [w]$ for $k = 1,\dots,s$;
- $b_k \in [w - h_k]$ for $k = 1,\dots,s-1$.

If we let $O_j := \{i_1,\dots,i_s\}$ and $\sigma_j(i_k) := t_{i_k} + h_k$, for $k = 1,\dots,s$, then the schedule (O_j,σ_j) is feasible, early start and $c(O_j,\sigma_j) = c(P_j)$.

If we go back to Example 1 and Fig. 4, we see that the red path represents the early start schedule where the courier serves the orders according to the sequence (1, 2, 3), which is case (i) in Example 1. The blue path represents instead the early start schedule where the courier serves the orders according to the sequence (1, 3, 2), which is case (ii).

It follows from above that each feasible early start schedule (O_j,σ_j) corresponds to a path P_j in $Q(N,A)$ starting from node $(j,0)$ and obeying to Remark 4, and vice versa every path P_j in $Q(N,A)$ starting from node $(j,0)$ and obeying to Remark 4 corresponds to a feasible early start schedule. Moreover, by construction, the cost $c(P_j)$ of path P_j is equal to $c(O_j,\sigma_j)$. Building upon Lemma 2, we may therefore state the following:

Theorem 5. *It is possible to reduce* OCAP *to the problem of finding a collection $\mathcal{P}$ of paths in the time extended network $Q(N,A)$ such that:*

(i) each path P_j leaves from the node $(j,0)$ associated with courier $j \in C$, and for each courier $j \in C$ there is at most one path leaving from node $(j,0)$;
(ii) for each order $i \in O$, the collection $\mathcal{P}$ is such that there exists at most one path $P_j \in \mathcal{P}$ that is entering $\delta^-(i)$, i.e., the layer of nodes associated with i;
(iii) the cost $c(\mathcal{P})$ is minimized: $c(\mathcal{P})$ is the sum of two terms: the cost $\sum_{j\in J} c(P_j)$ of the paths in $\mathcal{P}$ and the cost of rejections, namely, $\bar{c}$ times the number of orders not covered by any path in $\mathcal{P}$, i.e., nodes i such that $\bigcup_{j\in J} P_j \cap \delta^-(i) = \emptyset$.

4.4. An integer linear programming formulation for OCAP

Building upon Theorem 5, we provide an Integer Linear Programming formulation for OCAP. This program exploits two binary variables: y_a, which will be equal to 1 if and only if arc $a \in A$ belongs to some path $P_j \in \mathcal{P}$; w_i, which will be equal to 1 if and only if order $i \in O$ is rejected.

$$\min \sum_{i\in O} \bar{c}\cdot w_i + \sum_{a\in A} c_a \cdot y_a \tag{1}$$

$$\sum_{a\in\delta^+(j,0)} y_a \leq 1 \qquad \forall j \in C \tag{2}$$

$$\sum_{a\in\delta^+(i,h)} y_a \leq \sum_{a\in\delta^-(i,h)} y_a \qquad \forall i \in O, h \in [w], h \neq 0 \tag{3}$$

$$\sum_{a\in\delta^-(i)} y_a + w_i = 1 \qquad \forall i \in O \tag{4}$$

$$y_a \in \{0,1\} \qquad \forall a \in A \tag{5}$$

$$w_i \in \{0,1\} \qquad \forall i \in O \tag{6}$$

Constraints (2) impose that for each courier j there is at most one path leaving from node $(j,0)$ (see Theorem 5(i)). Constraints (3) impose flow conservation and that paths may start only at a node $(j,0)$ associated with some courier j (see again Theorem 5(i)). Constraints (4) impose that, for each order i, there is at most one path entering the layer of i and, in case there is none, force the corresponding variable w_i to be one.

We point out that the formulation above, which in the following we refer to as *F-formulation*, is a "flow-like" formulation, but for the constraints (4). In other words, if we skip constraints (4), then the constraints matrix is essentially a network matrix and therefore totally unimodular. Notice that we could write such a formulation – that is essentially a single commodity flow formulation – because once couriers start their work, they cannot be "distinguished" from each other. As soon as we impose constraints on that, say a given courier has to quit at a different time than T, or at time T she has to be back at her release location, we are not able to extend the above formulation and we need a multi-commodity flow one.

In Section 6, we will show that the F-formulation is indeed quite effective. We believe that keeping under control the size of the integer linear program, by restricting to early start schedules and thanks to the careful construction of the extended time network $Q(N,A)$, significantly contributes to the efficiency of the formulation.

5. Casting OCAP into a sequential decision process

The Offline Couriers Assignment Problem assumes a complete knowledge of the set of orders to be delivered. In this section, we sketch how to cast OCAP into the sequential decision process depicted in Fig. 3.

In our framework, each state s_k corresponds to an instance I^k of OCAP that is built on the base of the previous instance (state) and on the basis of the orders placed during the time step. We in particular build and solve a sequence of instances $I^0, I^1, \dots, I^T$ of OCAP where the index k of an instance I^k is consistent with the discretization defined in Section 3.2. Therefore, we think of a service as a finite set of time steps $0, 1, \dots, T$ each corresponding to a time instant $t^h = h \cdot \gamma$, for $h \in [T]$, where the base-step γ is small enough and in particular $\gamma \leq \alpha$ (we motivate this choice below). Then, we devote time γ to the solution of each instance I^k – we run an ILP solver on the *F-formulation* but any algorithm or heuristic for the solution of OCAP would work – provides decisions that contribute to the definition of the next instance I^{k+1}. Therefore, we build and solve a sequence of instances $I^0, I^1, \dots$ of OCAP, while adding at each time step the orders that have been placed in the last time window. More in detail, our approach is described by the following facts:

(i) We build a sequence of instances of OCAP $I^0, I^1, \dots, I^T$ such that each instance I_h, $h > 0$, is inductively built at time t^h upon instance I^{h-1}. We devote time γ to finding a solution $(\mathcal{O}^h, \sigma^h)$ to each instance I^h: therefore $(\mathcal{O}^h, \sigma^h)$ will be available at time $t^{h+1} = t^h + \gamma$. The solution $(\mathcal{O}^h, \sigma^h)$ will be the "reference" solution for the time window $[t^{h+1}, t^{h+2})$, as at time $t^{h+2} = t^{h+1} + \gamma$ a new solution $(\mathcal{O}^{h+1}, \sigma^{h+1})$ to I^{h+1} will be available.
(ii) Following Section 3.3, each instance I^h is defined by a tuple $(O^h, C, L^h, w, c, \bar{c}, \alpha, d)$. Note that while most values and parameters in the tuple are "static" and do not change from one instance to the other, the set of orders O^h will change e.g. because of placement times. However, the *attributes* of an order (pickup location, delivery location, start time, and average waiting time) will not change. To the contrary, while the set C of couriers does not change, the attributes of each courier, namely the release location p_j^h and the release time r_j^h, will very likely vary from one instance to the other. We therefore discuss below how O^h, p_j^h and r_j^h change.
(iia) For $h \geq 1$, the set O^h is built at time t^h by taking into account the orders placed in the time window $(t^{h-1}, t^h]$ and the decisions taken on orders placed before t^{h-1} by the solution $(\mathcal{O}^{h-1}, \sigma^{h-1})$ to the instance I^{h-1}. For orders in the latter class, there are four possible classes: the order has been delivered; the order has been rejected; the order is being "processed", i.e., according to

$(\mathcal{O}^{h-1}, \sigma^{h-1})$ the courier that has been assigned to that order is on her way to deliver it; the order is "not yet processed", even though $(\mathcal{O}^{h-1}, \sigma^{h-1})$ defines a courier and a schedule for that order. We then include in O^h, besides the orders placed in the time window $(t^{h-1}, t^h]$, only the orders not yet processed. It is therefore possible that, for some order in $O^h \cap O^{h-1}$, the solution $(\mathcal{O}^h, \sigma^h)$ overrides decisions previously taken by $(\mathcal{O}^{h-1}, \sigma^{h-1})$.

(*iib*) For $h \geq 1$, we infer the release location p_j^h and the release time r_j^h of a courier j for the instance I^h upon p_j^{h-1}, r_j^{h-1} and the solution $(\mathcal{O}^{h-1}, \sigma^{h-1})$. This, however, requires to fix where couriers spend possible idle times that arise according to $(\mathcal{O}^{h-1}, \sigma^{h-1})$: that is because we need to know how close or far they are from e.g. the pickup place of a new order placed in the last time window. So we need to fix *where* a courier on idle is spending her idle time: *we assume that couriers will always spend idle times at the delivery location of the last customer*. Under this assumption, there are two possible cases: either at time t^{h+1} the courier is moving to deliver an order $i \in O^{h-1}$ or she has not been moving for the entire time window $[t^h, t^{h+1})$ (that is why it is convenient to choose the time step γ not larger than the check-out time α). In the former case, we set the release location p_j^h as the delivery location $d(i)$ of order i and the release time r_j^h as the time at which j will complete the delivery of order i according to $(\mathcal{O}^{h-1}, \sigma^{h-1})$; in the latter case, we simply set $p_j^h := p_j^{h-1}$ and $r_j^h := r_j^{h-1}$.

(iii) We are left with describing the base instance I^0. As discussed above, we simply need to define the set of orders O^0 and, for each courier $j \in C$, the release location p_j^0 and release time r_j^0. The set of orders O^0 is made by all orders that have been placed by time t^0 (note that this set might be non-empty because customers may place orders before the shift starts); the release location p_j^0 and the release time r_j^0 are those that have been communicated at the beginning of the service, i.e., p_j and r_j.

Remark 6. When we build an instance I^h from instance I^{h-1} the time windows of each order are adjusted according to the advance of time, as no time window can start before the current time t^h (note that in the time extended network corresponding to I^h some layers will be shorter, as some nodes that would be in the past will be removed). We did not detail this aspect of the process above to avoid a too heavy notation.

6. Computational tests

In this section, we provide computational results on the solution of some instances of OCAP with the aim of validating both the quality of the F-formulation for the solution of OCAP and the sequential decision process presented in the previous section to cast OCAP into a dynamic setting.

Once again, we point out that when we solve a *single* instance OCAP the set of orders is fixed and known in advance; however when we cast OCAP into a dynamic setting and solve a *sequence* of instances of OCAP we must take into account the placement times of each order (as defined in Section 3.1): the placement time of an order will be a trigger for deciding which is the first instance that is aware of that order. Also note that the value of an optimal solution to an instance OCAP that takes into account the *entire* set of orders for a shift – i.e., no matter the placement times – provides a lower bound to that of any dynamic solution strategy that takes into account their placement times.

6.1. Base OCAP instances

We focus on the data already exploited in Section 3.1. In particular, we consider 33 dinner shifts for a day between January and March 2019 and derive 33 "base" instances $I_1, \dots, I_{33}$ of OCAP where we consider the *entire* set of orders that were placed for that service and the set of couriers that were hired for the service.

Recall that each instance of OCAP is indeed defined by a tuple $(O, C, L, w, c, \bar{c}, \alpha, d)$, see Section 3.3 for more details. We now provide a few details on the setting of $\mathcal{M}$ that are required for a precise definition of some values in the tuple.

Discretization. At dinner (the service we are interested in here) customers may ask for target times (i.e., delivery) between 19.00 and 22:30 However, couriers can start their work shift from 18:30 as they will use the first 30 min to get ready for the service and e.g. reach the location of a pickup from their release locations. In both cases, we set the base step γ, i.e., the length of a time step, equal to 5 min (see Section 3.2 for more details). Therefore, according to our definitions, the set of time steps is $[T] = \{0, 1, \dots, 42\}$.

Orders. Customers may place orders at any time, even outside $[T]$: this is the placement time. However, for each order i, they can only set a few target times $\theta_i' \in [T]$, namely those corresponding to :00 mins, :15 mins, :30 mins, and :45 mins. It follows that there are only 15 possible values for t_i'.

Width. The width w is equal to 15 as $w_l' = 3$ and $w_r' = 12$. Therefore, if θ_i is the true target time selected by the customer for order i, then the (discretized) time window for i is $W_i = \{t_i, t_i+1, \dots, t_i+15\}$, where t_i is the time step closest to θ_i.

Costs. The rejection cost $\bar{c}$ (recall that orders with delay larger than 60 min will always be rejected, see Section 1) is set equal to 10, while the non-decreasing delay cost function $c : [w] \mapsto \mathbb{Z}$ is defined as follows:

- $c(h) = 0$ for $h \in \{0, 1, \dots, 6\}$;
- $c(h) = 1$ for $h \in \{7, 8, 9\}$;
- $c(h) = 2$ for $h \in \{10, 11, 12\}$;
- $c(h) = 3$ for $h \in \{13, 14, 15\}$.

Travel Times and Check Out Time. Expected travel times are estimated through the package Geodesics.jl and assuming a speed of 30 km/h that is reasonable for riders in Rome (note that travel times will be then rounded to the closest time step according to the base step γ). The checkout time α is set equal to the base step γ, i.e., 5 min.

As a result, we got a set of 33 instances where the number of orders varies from 169 to 239; the number of couriers from 23 to 32; the ratio between the number of orders and the number of couriers from 6.47 to 7.62: in the following we refer to this value simply as the *ratio*. Details about these instances are given in the first four columns of Table 3 and the instances themselves can be found here: https://github.com/MatteoCosmi/Rome-Meal-Delivery-Instances.

6.2. Computational results for OCAP instances

We solved the 33 base instances discussed above by means of the F-formulation and the MIP solver of Gurobi 10.0.3, set with 8 threads. The experiments were carried out on a machine with a CPU 13th Gen Intel(R) Core(TM) i9-13900K and with 128 GB RAM. Computational details are given in the fifth and sixth columns of Table 3, where we report respectively the number of explored nodes in the Branch and Bound by the MIP solver and the time to find an optimal solution (in secs). In the last column we report the cost of the optimal solution and, when this cost is larger than 0, the number of orders that are delivered with a delay within $[15, 30)$ mins; the number of orders that are delivered with a delay within $[30, 45)$ mins; the number of orders that are delivered with a delay within $[45, 60)$ mins; the number of orders that have been rejected (for details on the cost function see Section 6.1).

Table 3
This table refers to the 33 base instances built upon real shifts from $\mathcal{M}$. For each instance, the table reports: the instance ID (Instance); the ratio between the number of orders and the number of couriers (Ratio); the number of orders (Orders); the number of couriers (Couriers); the number of explored nodes in the Branch and Bound (Nodes); the time to optimality (Time); the cost of the optimal solution (z^*): when positive, by (z_1, z_2, z_3, z_r) we respectively report: the number of orders with a delay within $[15, 30)$ mins (z_1); the number of orders with a delay within $[30, 45)$ mins (z_2); the number of orders with a delay within $[45, 60)$ mins (z_3); the number of orders that have been rejected (z_r).

Instance	Ratio	Orders	Couriers	Nodes	Time (secs)	z^*
$I1$	7.46	179	24	1	21.22	0
$I2$	7.47	239	32	1	33.92	0
$I3$	7.58	197	26	1	20.29	0
$I4$	7.52	188	25	1	12.21	0
$I5$	6.88	172	25	1	16.45	0
$I6$	7.10	213	30	1	26.05	0
$I7$	6.97	237	34	1	46.31	0
$I8$	7.44	186	25	1	24.31	0
$I9$	7.46	209	28	1	20.49	0
$I10$	6.90	200	29	1	32.39	0
$I11$	7.57	212	28	0	0.01	0
$I12$	7.42	178	24	1	27.68	0
$I13$	7.07	212	30	1	36.64	0
$I14$	7.58	182	24	1	13.57	0
$I15$	7.35	169	23	1157	357.56	0
$I16$	7.58	197	26	1	14.37	0
$I17$	7.48	202	27	1	25.38	0
$I18$	7.65	176	23	1	24.41	0
$I19$	7.52	173	23	1	14.98	0
$I20$	7.00	217	31	1	24.69	0
$I21$	6.54	183	28	1	7.97	0
$I22$	6.93	187	27	1	14.13	0
$I23$	7.48	172	23	1	11.5	0
$I24$	7.58	182	24	1	14.67	0
$I25$	7.06	219	31	1	15.67	0
$I26$	7.08	184	26	1	24.32	0
$I27$	7.48	187	25	0	0.01	0
$I28$	7.62	221	29	1	37.04	0
$I29$	6.59	191	29	1	16.71	0
$I30$	7.50	180	24	4853	125.48	3(3, 0, 0, 0)
$I31$	6.47	194	30	1	12.13	0
$I32$	7.58	197	26	1	29.14	0
$I33$	6.90	207	30	1	14.77	0

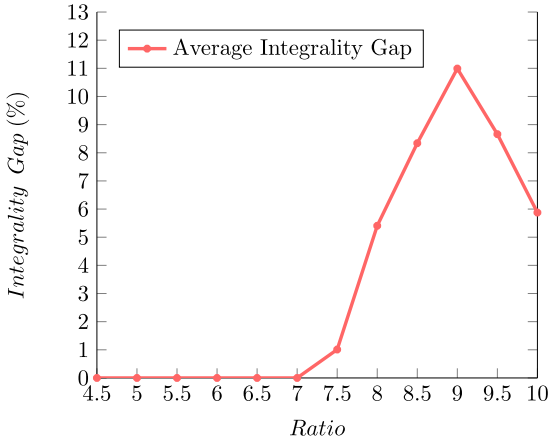

Fig. 5. The average value of the integrality gap for OCAP, for ratios in [4.5, 10].

We observe two main facts. First, all instances but three could be solved in less than a minute and the toughest in about six minutes. Then, the cost of the optimal solution is always 0 but for an instance where 3 orders are served within a delay between 15 and 30 min. We point out that at the time these instances were collected, $\mathcal{M}$ was using a manual dispatching of the orders that did not perform very well: therefore, in order to reduce delays too many couriers were hired. Note also that, since the cost function is non-negative, any feasible solution with cost 0 is trivially optimal, and so instances with optimal solution of value 0 are somehow "easier".

Therefore, in order to define more challenging instances for the F-formulation, we randomly removed from each instance in $\{I_1, \dots, I_{33}\}$ some couriers to impose larger ratios: namely, we set the ratio (approximately) equal to 8, 8.5, 9, 9.5 and 10 and therefore derived 165 additional instances of OCAP. These instances were indeed very interesting to the management of $\mathcal{M}$ as they allow to evaluate the trade-off between the cost of hiring less/more couriers and the QoS in terms of rejection and delays (ratios larger than 9 were anyhow considered too demanding for couriers).

For the sake of shortness we report in Table 4 detailed results only for ratios 8, 9, and 10. In particular, for each instance and each ratio, we report: z^*, the cost of the optimal solution with details on delays and rejections when this cost is positive (as we did for Table 3); $\lceil z^*_{RL} \rceil$, the cost (rounded up) of the optimal solution of the linear relaxation of the F-formulation; $Nodes$, the number of nodes that are explored in the branch and bound tree; $Time$ the computational time to find the optimal solution.

We observe that, as the ratio grows, the computation becomes indeed more challenging. Nevertheless, all instances could be solved to optimality, within 200 secs of computational time for ratio 8 (but for one instance) and within 2 h of computational time for ratio 9. Note also that z^* is always 0 when such is the value of z^*_{RL}. We think this is due to the good quality of the F-formulation, as it is also certified by the small values of $|z^* - z^*_{RL}|$ (the maximum is 7 and it comes for an instance with ratio 10) and by the *integrality gap* $\frac{z^* - \lceil z^*_{RL} \rceil}{z^*}$. The average value of the integrality gap is plotted in Fig. 5 for different ratios up to 10; note that we exploit the computations for ratios 8.5 and 9.5 as well as additional computations for ratios smaller than 8. Fig. 5 shows that the integrality gap stays below 11% and scales well when the ratio increases. One might be surprised that the maximum is attained at ratio 9 and the value is smaller at ratio 10. However, there is an explanation for that: when z^* is small, the value of the integrality gap is easily quite high. Indeed the average value for z^* is 23.55 for ratio 9 and 81.79 for ratio 10.

We finally point out that rejections are very uncommon and show up only for a few instances with ratio 9, which is quite high: a fact that was very much appreciated by the management of $\mathcal{M}$.

6.3. *Dynamic* OCAP *instances*

In the previous section, we showed that the F-formulation is quite effective for the solution of OCAP. However, there is a "catch" with OCAP, as we there assume all orders to be known in advance. In this section, we deal again with instances $I_1, \dots, I_{33}$, but cast them into the dynamic setting. We therefore consider the framework introduced in Section 5 and test the sequential decision process. For each instance $I_i, i = 1, \dots, 33$, we build a sequence of sub-instances $\{I_i^h, h = 0, \dots, T\}$ such that each instance I_i^h, $h > 0$, is inductively built at time t^h upon instance I_i^{h-1}, with $t^h = h \cdot \gamma$. We set $\gamma := 1$, i.e., we set the time distance between two sub-instances equal to one time step and therefore equal to 5 min.

We also point out that, in order to build I_i^h from I_i^{h-1}, we exploit the data on placement times collected by $\mathcal{M}$ for the shift corresponding to I_i: namely, we add the set of orders with placement time in the interval $(t^{h-1}, t^h]$. Finally, for each i, instance I_i^0 is built along the same lines discussed in Section 5 (see Item (iii)). This implies that the instances mimic the actual flow of orders during the considered shifts, representing a realistic example of a dynamic scenario.

6.4. *Computational results for dynamic* OCAP *instances*

For $i = 1, \dots, 33$, we solved the sequence of sub-instances $\{I_i^h, h = 0, \dots, T\}$ making again use of the MIP solver Gurobi 10.0.3. However,

Table 4
Testing high ratios. For instances $I_1, \dots, I_{33}$, the original ratio and then ratios 8, 9 and 10, we report: the cost of the optimal solution z^*: when positive, we report values (z_1, z_2, z_3, z_r), namely: the number of orders with a delay within $[15, 30)$ mins (z_1); the number of orders with a delay within $[30, 45)$ mins (z_2); the number of orders with a delay within $[45, 60)$ mins (z_3); the number of orders that have been rejected (z_r); the cost (rounded up) of the optimal solution to the linear relaxation of the F-formulation ($\lceil z^*_{RL} \rceil$), note that it is affected by Gurobi presolve phase; the number of nodes that are explored in the branch and bound tree ($Nodes$); the running time to optimal solution ($Time$).

Ratio	Original				8				9				10			
Instance	z^*	$\lceil z^*_{RL} \rceil$	Nodes	$Time$ (secs)	z^*	$\lceil z^*_{RL} \rceil$	$Nodes$	$Time$ (secs)	z^*	$\lceil z^*_{RL} \rceil$	$Nodes$	$Time$ (secs)	z^*	$\lceil z^*_{RL} \rceil$	$Nodes$	$Time$ (secs)
$I1$	0	0	1	21.22	0	0	1	17.71	4(4, 0, 0, 0)	2	2119	618.68	61(20, 13, 5, 0)	57	39 662	1863.22
$I2$	0	0	1	33.92	0	0	1	30.77	0	0	1	32.12	7(5, 1, 0, 0)	6	89 175	7786.76
$I3$	0	0	1	20.29	0	0	1	19.6	0	0	1	36.14	30(19, 4, 1, 0)	28	47 113	1761
$I4$	0	0	1	12.21	0	0	1	23.58	0	0	1	35.94	9(9, 0, 0, 0)	8	896	142.18
$I5$	0	0	1	16.45	0	0	1	14.38	35(17, 1, 2, 1)	32	25 952	1072.77	113(22, 14, 21, 0)	109	94 495	4964.01
$I6$	0	0	1	26.05	0	0	1	40.84	18(12, 0, 2, 0)	16	4380	739.86	114(28, 10, 22, 0)	108	121 482	10 139.29
$I7$	0	0	1	46.31	0	0	1	63.31	19(8, 4, 1, 0)	16	61 186	3218.84	71(26, 12, 7, 0)	67	358 287	94 791.65
$I8$	0	0	1	24.31	0	0	1	30.78	0	0	39	172.4	26(10, 8, 0, 0)	24	59 153	1472.43
$I9$	0	0	1	20.49	0	0	1	27.85	8(6, 1, 0, 0)	7	4	101.86	27(11, 5, 2, 0)	25	2927	1219.95
$I10$	0	0	1	32.39	18(3, 6, 1, 0)	17	6512	145.63	69(20, 5, 13, 0)	66	22 575	639.51	151(29, 22, 26, 0)	147	23 237	738.76
$I11$	0	0	0	0.01	0	0	1	31.66	0	0	1	65.21	23(15, 1, 2, 0)	20	147 160	18 500.35
$I12$	0	0	1	27.68	0	0	781	432.9	31(6, 5, 5, 0)	29	2868	503.99	102(25, 13, 17, 0)	98	37 600	2259.21
$I13$	0	0	1	36.64	7(5, 1, 0, 0)	5	7013	527.65	42(14, 3, 4, 1)	40	38 155	5277.31	158(28, 17, 32, 0)	152	785 175	73 366.57
$I14$	0	0	1	13.57	0	0	1	15.91	7(3, 2, 0, 0)	7	6411	112.46	55(12, 14, 5, 0)	51	3080	191.4
$I15$	0	0	1157	357.56	18(6, 3, 2, 0)	17	1296	198.43	55(13, 6, 10, 0)	53	1732	198.97	132(14, 14, 30, 0)	126	75 968	4233.68
$I16$	0	0	1	14.37	0	0	1	17.1	0	0	1	199.52	36(13, 4, 5, 0)	33	347 733	14 004.89
$I17$	0	0	1	25.38	0	0	1	48.6	48(23, 2, 7, 0)	45	5754	640.23	129(18, 21, 23, 0)	124	11 581	1517.1
$I18$	0	0	1	24.41	0	0	1	22.91	2(2, 0, 0, 0)	1	1	88.68	57(16, 10, 7, 0)	55	3722	507.77
$I19$	0	0	1	14.98	0	0	1	19.58	18(11, 2, 1, 0)	16	21 955	597.41	72(22, 10, 10, 0)	69	43 841	1580.93
$I20$	0	0	1	24.69	0	0	1	40.21	6(1, 1, 1, 0)	4	84 086	4453.37	56(16, 5, 10, 0)	52	356 138	21 723.42
$I21$	0	0	1	7.97	2(2, 0, 0, 0)	1	1954	156.22	27(6, 6, 3, 0)	25	1408	798.25	79(25, 9, 12, 0)	76	13 884	449.36
$I22$	0	0	1	14.13	0	0	1	44.38	19(11, 4, 0, 0)	17	15 628	567.96	76(27, 11, 9, 0)	72	48 990	1509.52
$I23$	0	0	1	11.5	0	0	1	13.72	0	0	1	34.68	37(14, 4, 5, 0)	34	40 649	683.62
$I24$	0	0	1	14.67	0	0	1	32.84	43(11, 4, 8, 0)	40	96 875	2717.82	122(21, 13, 25, 0)	117	114 075	6407.01
$I25$	0	0	1	15.67	0	0	1	43.6	24(8, 2, 4, 0)	21	145 129	5720.07	85(16, 12, 15, 0)	80	522 499	30 576.16
$I26$	0	0	1	24.32	0	0	1	57.84	28(14, 2, 0, 1)	25	35 587	1953.75	105(17, 14, 20, 0)	100	84 929	8100.82
$I27$	0	0	0	0.01	0	0	1	21.3	3(3, 0, 0, 0)	2	1	46.67	18(8, 2, 2, 0)	17	417	741.06
$I28$	0	0	1	37.04	0	0	1	42.92	4(4, 0, 0, 0)	3	14 618	2244.99	71(18, 10, 11, 0)	67	58 977	14 309.27
$I29$	0	0	1	16.71	2(2, 0, 0, 0)	1	546	162.21	54(16, 4, 10, 0)	51	150 348	3339.69	134(21, 19, 25, 0)	129	273 888	18 594.25
$I30$	3(3, 0, 0, 0)	2	4853	125.48	37(18, 5, 3, 0)	35	865	107.03	96(22, 6, 14, 2)	92	18 207	666.12	198(15, 24, 45, 0)	191	73 107	3890
$I31$	0	0	1	12.13	0	0	1	118.58	22(9, 5, 1, 0)	19	4457	798.63	119(19, 14, 24, 0)	114	8716	1461.57
$I32$	0	0	1	29.14	0	0	39	146.08	50(19, 8, 5, 0)	47	180 506	7144.04	124(26, 10, 26, 0)	119	64 146	7710.21
$I33$	0	0	1	14.77	3(3, 0, 0, 0)	2	1427	70.78	45(12, 9, 5, 0)	42	4368	400.43	102(20, 17, 16, 0)	97	281 018	10241.59
Mean	0.09	0.06	183	33.51	2.64	2.36	619.91	84.45	23.55	21.73	28 616.82	1370.86	81.79	77.94	128 294.55	11 134.52

for each instance I_i^h, we set a time limit of 5 min for the overall computation, i.e., we stop the computation if it is still running when reaching the next time step. Note that therefore, it might happen that an instance I_i^h is not solved to optimality. In this case, we simply use the best solution found by the solver so far. Note also that the solution found for I_i^h will be used as a warm start for solving I_i^{h+1} (we skip technical details).

The results on the 33 base instances are shown in the first 4 columns of Table 5 (the values in the other columns are discussed below). For each instance $I_1, \dots, I_{33}$ we report: z^D, the cost of the solution provided by the sequential decision process (D stands for dynamic); z^*, the cost of the optimal solution to the same instance but in an offline setting (these are the same values reported in Table 4: recall that $z^D \geq z^*$); $Opt.\ Solver$, the percentage of times that in the sequential decision process the solver was able to find an optimal solution within the time limit of 5 min.

We point out that the average value of $Opt.\ Solver$ is about 94%, meaning that the 5-minutes time limit is not too severe. Also for 17 out of 33 instances, $z^D = z^*$. These results seem to suggest that it is indeed possible to combine the solution of OCAP with the sequential decision process. We decided however to stress again our model by reducing the number of couriers, as we did in Section 6.2. We, therefore, consider again the 165 additional instances of OCAP defined there and cast them into the dynamic setting by means of the same technique described in Section 6.3. Results for ratios 8, 9, and 10 are reported in the other columns of Table 5.

Note that also when the ratio increases the solver is able on average to find an optimal solution within the time limit (one time step) for more than 91% of the calls. As for the values of z^D and z^*, we plot in Fig. 6 their average values for different ratios (we again exploit the results for instances with ratio 8.5 and 9.5 as well as additional computations for ratios smaller than 8). Note that for large enough values of the ratio, the value $\frac{z^D}{z^*}$ is monotonically decreasing: it is 6.86 for ratio 8; 3.95 for ratio 8.5; 2.50 for ratio 9; 1,87 for ratio 9.5; 1.5 for ratio 10. Overall, the results in this section show that our solution algorithm to OCAP scales pretty well to the case where we cast it into a sequential decision process.

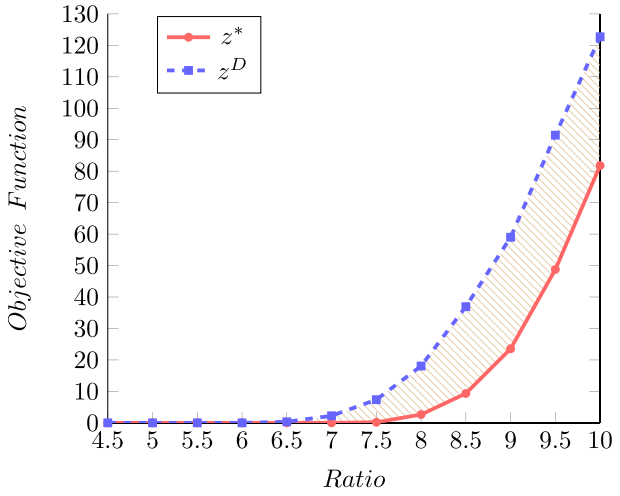

Fig. 6. The solid-red and dashed-blue lines represent respectively the average values of the offline solutions to OCAP and the average values of the solutions found by the sequential decision approach, for ratios ranging from 4.5 to 10.

7. Conclusions

We investigate some optimization models for meal delivery stemming from a collaboration with $\mathcal{M}$, a meal delivery company operating in Rome. The focus of this collaboration was the design of optimization models for dispatching orders to couriers so as to avoid as many as possible delays and rejections in a setting where a large part of

Table 5
Testing the sequential decision process. For instances $I_1, \ldots, I_{33}$, the original ratio and then ratios 8, 9 and 10, we report: the cost of the solution provided by the sequential decision process (z^D), the optimal solution to the off-line problem (z^*), which is the same value reported in Table 4; the percentage of times the solver was able to find an optimal solution within the time limit of 5 min (*Opt. Solver*). Note that for both z^D and z^*, when they are positive, we report values (z_1, z_2, z_3, z_r), namely: the number of orders with a delay within $[15, 30)$ mins (z_1); the number of orders with a delay within $[30, 45)$ mins (z_2); the number of orders with a delay within $[45, 60)$ mins (z_3); the number of orders that have been rejected (z_r).

Ratio	Original			8			9			10		
Instance	z^D	z^*	Opt. Solver (%)	z^D	z^*	Opt. Solver (%)	z^{RH}	z^*	Opt. Solver (%)	z^D	z^*	Opt. Solver (%)
$I1$	0	0	80.00	5(3, 1, 0, 0)	0	94.12	7(5, 1, 0, 0)	0	95.45	33(18, 6, 1, 0)	4(4, 0, 0, 0)	91.14
$I2$	0	0	100.00	0	0	100.00	8(6, 1, 0, 0)	0	89.47	14(6, 1, 2, 0)	0	94.87
$I3$	0	0	100.00	0	0	100.00	0	0	100.00	9(9, 0, 0, 0)	0	97.67
$I4$	1(1, 0, 0, 0)	0	90.91	1(1, 0, 0, 0)	0	90.91	5(5, 0, 0, 0)	0	93.75	8(6, 1, 0, 0)	0	95.00
$I5$	0	0	100.00	2(0, 1, 0, 0)	0	90.32	26(10, 5, 2, 0)	12(8, 2, 0, 0)	94.74	51(19, 7, 6, 0)	35(17, 1, 2, 1)	94.44
$I6$	0	0	85.00	13(6, 2, 1, 0)	0	95.24	39(16, 4, 5, 0)	6(3, 0, 1, 0)	92.65	58(30, 5, 6, 0)	18(12, 0, 2, 0)	91.67
$I7$	0	0	100.00	12(8, 2, 0, 0)	0	100.00	36(15, 9, 1, 0)	0	100.00	79(21, 17, 8, 0)	19(8, 4, 1, 0)	99.06
$I8$	2(0, 1, 0, 0)	0	72.73	7(5, 1, 0, 0)	0	88.46	13(9, 2, 0, 0)	0	92.86	19(11, 4, 0, 0)	0	95.65
$I9$	0	0	100.00	0	0	100.00	1(1, 0, 0, 0)	0	100.00	10(4, 3, 0, 0)	8(6, 1, 0, 0)	87.80
$I10$	0	0	100.00	27(6, 3, 5, 0)	18(3, 6, 1, 0)	88.24	35(10, 5, 5, 0)	30(11, 5, 3, 0)	88.52	85(16, 10, 13, 1)	69(20, 5, 13, 0)	85.88
$I11$	0	0	100.00	0	0	88.89	1(1, 0, 0, 0)	0	95.24	8(8, 0, 0, 0)	0	95.00
$I12$	1(1, 0, 0, 0)	0	92.59	31(16, 3, 3, 0)	0	94.34	47(19, 5, 6, 0)	11(6, 1, 1, 0)	93.06	63(17, 11, 8, 0)	31(6, 5, 5, 0)	91.86
$I13$	5(3, 1, 0, 0)	0	96.15	48(16, 7, 6, 0)	7(5, 1, 0, 0)	96.05	75(28, 1, 15, 0)	21(10, 4, 1, 0)	94.25	98(20, 7, 18, 1)	42(14, 3, 4, 1)	95.41
$I14$	0	0	93.75	2(2, 0, 0, 0)	0	85.19	21(12, 3, 1, 0)	0	88.52	43(21, 8, 2, 0)	7(3, 2, 0, 0)	87.84
$I15$	15(8, 2, 1, 0)	0	93.02	37(11, 7, 4, 0)	18(6, 3, 2, 0)	90.74	59(6, 10, 11, 0)	34(8, 7, 4, 0)	91.55	77(13, 9, 12, 1)	55(13, 6, 10, 0)	91.55
$I16$	0	0	89.47	1(1, 0, 0, 0)	0	89.19	32(15, 1, 5, 0)	0	92.21	48(17, 5, 7, 0)	0	90.80
$I17$	2(2, 0, 0, 0)	0	96.30	32(14, 3, 4, 0)	0	94.12	49(23, 4, 6, 0)	4(4, 0, 0, 0)	94.12	94(24, 12, 12, 1)	48(23, 2, 7, 0)	95.37
$I18$	2(2, 0, 0, 0)	0	93.94	14(6, 1, 2, 0)	0	97.56	26(12, 4, 2, 0)	0	94.83	54(15, 12, 5, 0)	2(2, 0, 0, 0)	95.83
$I19$	5(5, 0, 0, 0)	0	100.00	11(7, 2, 0, 0)	0	100.00	33(16, 4, 3, 0)	4(2, 1, 0, 0)	94.92	64(18, 8, 10, 0)	18(11, 2, 1, 0)	95.95
$I20$	0	0	100.00	5(2, 0, 1, 0)	0	92.68	21(9, 0, 4, 0)	0	98.36	56(18, 7, 8, 0)	6(1, 1, 1, 0)	94.68
$I21$	0	0	100.00	9(6, 0, 1, 0)	2(2, 0, 0, 0)	90.91	14(7, 2, 1, 0)	8(4, 2, 0, 0)	84.85	40(8, 4, 8, 0)	27(6, 6, 3, 0)	84.78
$I22$	6(6, 0, 0, 0)	0	100.00	32(18, 4, 2, 0)	0	96.83	55(20, 7, 7, 0)	5(5, 0, 0, 0)	96.20	82(18, 11, 14, 0)	19(11, 4, 0, 0)	94.68
$I23$	1(1, 0, 0, 0)	0	85.71	0	0	75.00	11(7, 2, 0, 0)	0	88.89	21(7, 4, 2, 0)	0	89.58
$I24$	20(8, 3, 2, 0)	0	96.23	35(10, 5, 5, 0)	0	98.31	82(16, 6, 18, 0)	19(8, 1, 3, 0)	93.55	112(20, 16, 20, 0)	43(11, 4, 8, 0)	93.20
$I25$	0	0	91.67	16(4, 3, 2, 0)	0	92.31	32(13, 2, 5, 0)	0	95.31	68(24, 7, 10, 0)	24(8, 2, 4, 0)	91.21
$I26$	0	0	100.00	9(9, 0, 0, 0)	0	92.45	31(18, 5, 1, 0)	0	94.44	71(24, 5, 9, 1)	28(14, 2, 0, 1)	93.41
$I27$	0	0	100.00	1(1, 0, 0, 0)	0	84.62	6(4, 1, 0, 0)	0	89.47	8(4, 2, 0, 0)	3(3, 0, 0, 0)	88.00
$I28$	8(6, 1, 0, 0)	0	85.00	13(8, 1, 1, 0)	0	86.54	43(17, 4, 6, 0)	0	88.31	68(22, 11, 8, 0)	4(4, 0, 0, 0)	89.69
$I29$	2(2, 0, 0, 0)	0	96.43	49(19, 6, 6, 0)	2(2, 0, 0, 0)	92.31	92(21, 13, 15, 0)	31(9, 2, 6, 0)	93.75	117(20, 12, 21, 1)	54(16, 4, 10, 0)	94.17
$I30$	41(20, 6, 3, 0)	3(3, 0, 0, 0)	95.31	76(24, 11, 10, 0)	37(18, 5, 3, 0)	93.26	107(24, 16, 17, 0)	63(20, 11, 7, 0)	93.62	146(19, 23, 27, 0)	96(22, 9, 14, 2)	93.94
$I31$	0	0	100.00	15(9, 3, 0, 0)	0	90.91	35(13, 8, 2, 0)	6(3, 0, 1, 0)	90.77	48(16, 10, 4, 0)	22(9, 5, 1, 0)	94.52
$I32$	1(1, 0, 0, 0)	0	100.00	39(16, 4, 5, 0)	0	95.45	87(21, 12, 14, 0)	27(12, 3, 3, 0)	95.51	104(22, 11, 20, 0)	50(19, 8, 5, 0)	94.90
$I33$	27(13, 7, 0, 0)	0	94.12	43(24, 8, 1, 0)	3(3, 0, 0, 0)	92.42	90(20, 11, 16, 0)	27(11, 5, 2, 0)	94.25	91(20, 10, 17, 0)	45(12, 9, 5, 0)	94.32
Mean	4.21	0.09	94.80	18.06	2.64	92.65	59.00	23.55	92.85	122.73	81.79	91.25

customers' orders are known in advance. This took us to design a sequential decision process based on a rolling horizon approach where we do not try to anticipate future demands. At the core of this approach, there is the solution through integer linear programming of a fully deterministic optimization problem, the Offline Couriers Assignment Problem (OCAP), and, in particular, a suitable "flow-like" formulation for the latter. This formulation exploits both the hypothesis that couriers (once they start their shift) cannot be "distinguished" from each other and some monotonicity of the delay cost function. We validated both the use of this formulation to solve OCAP, and the solution of OCAP itself within the sequential decision process, through some computational tests on real instances collected on the ground, that we make available to the scientific community.

We now discuss a few generalizations of our model, part of which we plan to investigate in the future. First, we point out that our model can easily handle the case where we assume that each order has its own width and its own rejection cost, which in practice is a simple way for providing higher QoS to valuable customers (e.g. the width could be smaller and the rejection cost higher for their orders).

On the contrary, there are other natural generalizations that cannot be taken without loss of generality for our model. (*i*) We may also handle non-regular delay cost function, i.e., delay cost-function $c : [w] \mapsto \mathbb{Z}$ such that it is not always the case that $c(h) \leq c(h+1)$; however, in this case, we expect to pay some computational cost as Lemma 2 does not need to hold and the size of the network used by the F-formulation could be larger. (*ii*) Our model exploits the hypothesis that couriers start their work shift at different times and locations but they must be available until the end of the service, without any constraint on the time and location of their last delivery. Therefore, once a courier starts her work shift, she cannot be "distinguished" from other active couriers and that is why the F-formulation is essentially a *single-commodity* flow formulation. However, if the latter constraints are to be taken into account by the model, then we need a *multi-commodity* formulation, and, once again, we expect to pay higher computational costs. (*iii*) Finally, a natural extension of the Offline Couriers Assignment Problem is that of considering orders' aggregation (e.g. a courier could pick different orders from the same restaurant), but this was not considered in this research because it conflicts with the maximization of QoS which was the main focus of $\mathcal{M}$.

CRediT authorship contribution statement

Matteo Cosmi: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Data curation. **Gianpaolo Oriolo:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Methodology, Investigation, Formal analysis, Conceptualization. **Veronica Piccialli:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Methodology, Investigation, Data curation, Conceptualization. **Paolo Ventura:** Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Methodology, Investigation, Formal analysis, Data curation.

Declaration of competing interest

The authors wish to confirm that there are no known conflicts of interest associated with the publication Efficient Courier Assignment in Meal Delivery via Integer Programming by Matteo Cosmi, Gianpaolo Oriolo, Veronica Piccialli and Paolo Ventura.

Data availability

The data are shared at the link https://github.com/MatteoCosmi/Rome-Meal-Delivery-Instances.

References

[1] Cosmi Matteo, Oriolo Gianpaolo, Piccialli Veronica, Ventura Paolo. Single courier single restaurant meal delivery (without routing). Oper Res Lett 2019;47(6):537–41. http://dx.doi.org/10.1016/j.orl.2019.09.007.

[2] Cordeau Jean-François, Laporte Gilbert, Potvin Jeana Yves, Savelsbergh Martin. Chapter 7 Transportation on demand. Handbooks Oper Res Management Sci 2007;14(C):429–66. http://dx.doi.org/10.1016/S0927-0507(06)14007-4, URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-77950475013&doi=10.1016%2fS0927-0507%2806%2914007-4&partnerID=40&md5=0d41e4c655b72008797313455c0a19ed.

[3] Cant Callum. Riding for deliveroo: Resistance in the new economy. Wiley; 2019.

[4] Martínez-Sykora Antonio, McLeod Fraser, Cherrett Tom, Friday Adrian. Exploring fairness in food delivery routing and scheduling problems. Expert Syst Appl 2024;240:122488. http://dx.doi.org/10.1016/j.eswa.2023.122488.

[5] Cordeau Jean-François, Laporte Gilbert. The dial-a-ride problem: models and algorithms. Ann Oper Res 2007;153:29–46. http://dx.doi.org/10.1007/s10479-007-0170-8.

[6] Molenbruch Yves, Braekers Kris, Caris An. Typology and literature review for dial-a-ride problems. Ann Oper Res 2017;259:295–325. http://dx.doi.org/10.1007/s10479-017-2525-0.

[7] Ho Sina C, Szeto WY, Kuo Yong-Hong, Leung Jannya MY, Petering Matthew, Tou Terencea WH. A survey of dial-a-ride problems: Literature review and recent developments. Transp Res B 2018;111:395–421. http://dx.doi.org/10.1016/j.trb.2018.02.001.

[8] Gökay Sevket, Heuvels Andreas, Krempels Karl-Heinz. A high-level category survey of dial-a-ride problems. In: Proceedings of the 5th international conference on vehicle technology and intelligent transport systems. SciTePress; 2019, p. 594–600. http://dx.doi.org/10.5220/0007801605940600.

[9] Ulmer Marlina W, Goodson Justina C, Mattfeld Dirka C, Thomas Barretta W. On modeling stochastic dynamic vehicle routing problems. EURO J Transp Logist 2020;9(2):100008. http://dx.doi.org/10.1016/j.ejtl.2020.100008.

[10] Klein Vienna, Steinhardt Claudius. Dynamic demand management and online tour planning for same-day delivery. European J Oper Res 2022. http://dx.doi.org/10.1016/j.ejor.2022.09.011.

[11] Klapp Mathiasa A, Erera Alan, Toriello Alejandro. The dynamic dispatch waves problem for same-day delivery. European J Oper Res 2018;271(2):519–34. http://dx.doi.org/10.1016/j.ejor.2018.05.032.

[12] Klapp Mathiasa A, Erera Alan, Toriello Alejandro. The one-dimensional dynamic dispatch waves problem. Transp Sci 2018;52(2):402–15. http://dx.doi.org/10.1287/trsc.2016.0682.

[13] Ulmer Marlina W, Thomas Barretta W, Mattfeld Dirka C. Preemptive depot returns for dynamic same-day delivery. EURO J Transp Logist 2019;8(4):327–61. http://dx.doi.org/10.1007/s13676-018-0124-0.

[14] Voccia Stacya A, Campbell Anna Melissa, Thomas Barretta W. The same-day delivery problem for online purchases. Transp Sci 2019;53(1):167–84. http://dx.doi.org/10.1287/trsc.2016.0732.

[15] Stroh Alexandera M, Erera Alan, Toriello Alejandro. Tactical design of same-day delivery systems. Manage Sci 2022;68(5):3444–63. http://dx.doi.org/10.1287/mnsc.2021.4041.

[16] Chen Xinwei, Ulmer Marlina W, Thomas Barretta W. Deep Q-learning for same-day delivery with vehicles and drones. European J Oper Res 2022;298(3):939–52. http://dx.doi.org/10.1016/j.ejor.2021.06.021.

[17] Côté Jean-François, Alves de Queiroz Thiago, Gallesi Francesco, Iori Manuel. A branch-and-regret algorithm for the same-day delivery problem. Transp Res E 2023;177:103226. http://dx.doi.org/10.1016/j.tre.2023.103226.

[18] Pillac Victor, Gendreau Michel, Guéret Christelle, Medaglia Andrésa L. A review of dynamic vehicle routing problems. European J Oper Res 2013;225(1):1–11. http://dx.doi.org/10.1016/j.ejor.2012.08.015.

[19] Reyes Damian, Erera Alan, Savelsbergh Martin, Sahasrabudhe Sagar, O'Neil Ryan. The meal delivery routing problem. Optimization-Online 2018. URL: http://www.optimization-online.org/DB_HTML/2018/04/6571.html.

[20] Yildiz Baris, Savelsbergh Martin. Provably high-quality solutions for the meal delivery routing problem. Transp Sci 2019;53(5):1372–88. http://dx.doi.org/10.1287/trsc.2018.0887.

[21] Steever Zachary, Karwan Mark, Murray Chase. Dynamic courier routing for a food delivery service. Comput Oper Res 2019;107:173–88. http://dx.doi.org/10.1016/j.cor.2019.03.008.

[22] Ulmer Marlina W, Thomas Barretta W, Campbell Anna Melissa, Woyak Nicholas. The restaurant meal delivery problem: Dynamic pickup and delivery with deadlines and random ready times. Transp Sci 2020;55(1):75–100. http://dx.doi.org/10.1287/trsc.2020.1000.

[23] Auad Ramon, Erera Alan, Savelsbergh Martin. Courier satisfaction in rapid delivery systems using dynamic operating regions. Omega 2023;121:102917. http://dx.doi.org/10.1016/j.omega.2023.102917.

[24] Xue Guiqin, Wang Zheng, Wang Guan. Optimization of rider scheduling for a food delivery service in O2O business. J Adv Transp 2021;2021. http://dx.doi.org/10.1155/2021/5515909.

[25] Liao Wenzhu, Zhang Liuyang, Wei Zhenzhen. Multi-objective green meal delivery routing problem based on a two-stage solution strategy. J Clean Prod 2020;258:120627. http://dx.doi.org/10.1016/j.jclepro.2020.120627.

[26] Bozanta Aysun, Cevik Mucahit, Kavaklioglu Can, Kavuk Eraya M, Tosun Ayse, Sonuc Sibela B, et al. Courier routing and assignment for food delivery service using reinforcement learning. Comput Ind Eng 2022;164:107871. http://dx.doi.org/10.1016/j.cie.2021.107871.

[27] Ausseil Rosemonde, Ulmer Marlina W, Pazour Jennifera A. Online acceptance probability approximation in peer-to-peer transportation. Omega 2024;123:102993. http://dx.doi.org/10.1016/j.omega.2023.102993.

[28] Böhm Martin, Megow Nicole, Schlöter Jens. Throughput scheduling with equal additive laxity. In: Calamoneri Tiziana, Coró Federico, editors. Algorithms and complexity. Springer International Publishing; 2021, p. 130–43.

[29] Cosmi Matteo, Nicosia Gaia, Pacifici Andrea. Lower bounds for a meal pickup-and-delivery scheduling problem. In: Proceedings of the 17th cologne-twente workshop on graphs and combinatorial optimization. 2019, p. 33–6, URL: https://foreman.virt.dacs.utwente.nl/~ctw/CTW2019ProceedingsFinal.pdf.

[30] Cosmi Matteo, Nicosia Gaia, Pacifici Andrea. Scheduling for last-mile meal-delivery processes. In: IFAC-papersOnLine. IFAC-MIM conference 2019, vol. 52, no. 13, 2019, p. 511–6. http://dx.doi.org/10.1016/j.ifacol.2019.11.117.

[31] Agnetis Alessandro, Cosmi Matteo, Nicosia Gaia, Pacifici Andrea. Two is better than one? Order aggregation in a meal delivery scheduling problem. Comput Ind Eng 2023;109514. http://dx.doi.org/10.1016/j.cie.2023.109514.

[32] Joshi Manas, Singh Arshdeep, Ranu Sayan, Bagchi Amitabha, Karia Priyank, Kala Puneet. Batching and matching for food delivery in dynamic road networks. In: 2021 IEEE 37th international conference on data engineering. Los Alamitos, CA, USA: IEEE Computer Society; 2021, p. 2099–104. http://dx.doi.org/10.1109/ICDE51399.2021.00207.

[33] Joshi Manas, Singh Arshdeep, Ranu Sayan, Bagchi Amitabha, Karia Priyank, Kala Puneet. FoodMatch: Batching and matching for food delivery in dynamic road networks. ACM Trans Spatial Algorithms Syst 2022;8(1). http://dx.doi.org/10.1145/3494530.

[34] Böhm Martin, Megow Nicole, Schlöter Jens. Throughput scheduling with equal additive laxity. Oper Res Lett 2022;50(5):463–9. http://dx.doi.org/10.1016/j.orl.2022.06.007.

[35] Soeffker Ninja, Ulmer Marlina W, Mattfeld Dirka C. Stochastic dynamic vehicle routing in the light of prescriptive analytics: A review. European J Oper Res 2022;298(3):801–20. http://dx.doi.org/10.1016/j.cie.2023.109514.

[36] Garey MR, Johnson Davida S. Computers and intractability: A guide to the theory of NP-ompleteness. W. H. Freeman; 1979.