Assigning Orders to Couriers in Meal Delivery via Integer Programming

Matteo Cosmi[*] Gianpaolo Oriolo[†] Veronica Piccialli[‡]
Paolo Ventura[§]

Abstract

We investigate some optimization models for meal delivery that stem from a collaboration with an Italian company mainly operating in Rome. The focus of this company is on top-end customers, and the company pursues high Quality of Service through a careful management of delays. We then design optimization models and algorithms for dispatching orders to couriers to avoid as much as possible delays and cancellations. The solution approach is based on the iterative solution of fully deterministic optimization sub-problems that are solved through integer programs exploiting a suitable "flow-like" formulation. We validate both the approach and the integer programming formulation through some computational tests on some real instances collected on the ground during our collaboration. We make these instances available to the scientific community and discuss a few insights on the meal delivery market in the area of Rome that may be of interest to other research groups.

Keywords: Integer Programming, Meal Delivery, Time Extended Network, FLow-like Formulation

[*]Luxembourg Centre for Logistics and Supply Chain Management (LCL), University of Luxembourg, 6, rue Richard Coudenhove-Kalergi, L-1359 Luxembourg, Luxembourg. `matteo.cosmi@uni.lu`

[†]Corresponding Author, Dipartimento di Ingegneria Civile e Ingegneria Informatica, Università degli Studi di Roma "Tor Vergata", via del Politecnico 1, 00133 Roma, Italy. `oriolo@disp.uniroma2.it`,

[‡]Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Università degli Studi di Roma "La Sapienza", via Ariosto 25, 00185, Roma, Italy. `veronica.piccialli@uniroma1.it`

[§]Istituto di analisi dei sistemi ed informatica "Antonio Ruberti", CNR, via dei Taurini 19, 00185, Roma, Italy. `paolo.ventura@iasi.cnr.it`

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, GrubHub, Just-Eat or Uber Eats. This increasing interest for the meal delivery logistic is also witnessed by a number of 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 as $\mathcal{M}$ mainly operating in the area of Roma: this work indeed steams from a collaboration with this company. $\mathcal{M}$ focuses on top-end customers and pursues high Quality of Service in terms of delays in delivering the orders, while obtaining positive commitment of couriers. To this aim, couriers are hired before the shift and only a small part of their salary depends on the orders that are then delivered during the shift. Also the assignment of orders to couriers is centralized and couriers cannot refuse an order that has been assigned to them[1]. As for the management of delays, orders with delay larger than 60 minutes are cancelled, and $\mathcal{M}$ gives a discount voucher to customers that experience a delay, or a cancellation.

Given these premises, the main target of our collaboration with $\mathcal{M}$ was the design of an optimization solutions for dispatching orders to couriers as to avoid too as much possible delays and cancellations. In order to shape our solution approach the first step we took was to analyze some data collected on the ground by $\mathcal{M}$: these data and a few figures concerning customer demand for meal delivery in the area of Rome will be extensively discussed in Section 2. We in particular observed that most of the meal orders to $\mathcal{M}$ are placed by customers highly in advance, and only a very small fraction of customers requires a "best-effort" delivery, i.e., as soon as possible (once again recall that the focus of the company is on top-end customers and they are more likely to book largely in advance). For this reason, in agreement with the management of $\mathcal{M}$, we chose to develop a solution strategy that is based on the iterative solution of a fully deterministic optimization problem where we assume a complete knowledge of the orders to be served. Namely, we define a fully deterministic optimization problem, that we called the *Offline Couriers Assignment Problem* (OCAP), and solve OCAP in a *rolling horizon* fashion where we update the set of orders to be served as to follow the

[1]Note that according to $\mathcal{M}$ business model orders are therefore assigned to couriers on a "push" base, while other models are "pull" based and couriers will choose orders.

insurgence of new demands (and possibly delays and failures with respect to current orders): this approach is presented in Section 2.1 and then detailed in Section 3.1 and 4.

A solution to OCAP provides an assignment of orders to couriers as to minimize a suitable cost function, that takes into account delays and cancellations, without taking into account any cost related to the number of couriers, as they are hired before the shift, and to the cost of the routing, as each courier will autonomously choose her route. The definition of OCAP was suggested by our collaboration with $\mathcal{M}$, nevertheless our ambition here is that of stating a general problem that could be of interest also in other settings. Therefore, we devote Section 3 to discuss the main assumptions behind OCAP, Section 3.1 to a formal definition of the problem and only in Section 3.2 we tailor OCAP to the setting of $\mathcal{M}$.

The rolling horizon approach that allows to exploiting OCAP in an "dynamic" setting is discussed in Section 4. Namely, for a given shift, we define a sequence of instances of OCAP such that each instance inherits from the previous one a suitable set of orders and side constraints, while adding new orders that have been placed by customers.

It follows from above that the core of our solution strategy for dispatching orders to couriers (both in an offline and a dynamic scenario) is the solution to OCAP. We know from Cosmi, Oriolo, et al. (2019) that (the decision version of) OCAP is NP-HARD already for the case of a single courier. We therefore approach OCAP by Integer Linear Programming and in particular propose a "flow-like" formulation that exploits a suitable time extended network: both the time extended network and the Integer Linear Programming formulation are presented in Section 5. We keep under control the size of the extended network (and of the linear program) by exploiting some monotonicity property of the cost function, that is discussed in Section 5.1.

We devoted the last part of our collaboration with $\mathcal{M}$ to validate our solution strategy: Section 6, reports on that. First, we present a set of 33 real instances collected on the ground in the period ranging from January to March 2018[2] that we make available to the scientific community. We exploit these instances to validate both the use of the flow-like formulation to solve OCAP and the rolling horizon approach to deal with the dynamic setting. In all experiments, our solutions showed very good performances, proving that, on the one hand, the flow-like formulation is very effective to deal with OCAP, on the other hand, the rolling horizon approach fits to the dynamic setting.

[2]Note that the numbers in these instances (number of orders, numbers of couriers etc.) are not representative of the current volumes handled by $\mathcal{M}$.

Last but not least, building again upon OCAP, we ran some experiments to estimate which is the "right" ratio between the number of orders and the number of couriers in a shift, as to avoid delays and cancellations (at least for the metropolitan area of Rome). We were indeed able to prove to the management of $\mathcal{M}$ that, by means of our solution approach, they could reduce the number of couriers hired in a shift without affecting the quality of service (before our collaboration orders were dispatched almost manually and, as a matter of fact, that did not work very well).

1.1 Related Literature

The interest in meal delivery logistics is witnessed by the increasing literature in optimization journals starting in 2018. In one of the first contributions Reyes et al. (2018) define the "Meal Delivery Problem" and propose a deterministic dynamic model to deal with dispatching orders to couriers, finding routes for them and also deal with the problem of dimensioning the fleet of drivers. However, most of the forthcoming literature on variants of the meal delivery problem focuses on stochastic approaches to deal with different sources of uncertainties: the order placement, the fluctuations of customer demand over time and space, and the waiting time at the restaurant for couriers (see e.g. Paul et al. (2020); Ulmer, Thomas, et al. (2020); Xue, Z. Wang, and G. Wang (2021); Chu et al. (2021); Ulmer, Thomas, et al. (2020); Steever, Karwan, and Murray (2019); Liao, Zhang, and Wei (2020); Bozanta et al. (2022); Chen et al. (2021); Zehtabian, Larsen, and Wøhlk (2022)).

Moving to complexity issues Cosmi, Oriolo, et al. (2019) and Böhm, Megow, and Schlöter (2021) analyze some basic offline optimization models that are related to the models discussed in this paper (see Section 3.1). In the simplest model, a single restaurant entrusts a single courier to deliver meals to customers who place orders at the restaurant; Cosmi, Nicosia, and Pacifici (2019a); Cosmi, Nicosia, and Pacifici (2019b) and Agnetis et al. (2021) elaborate on that model and propose exact solution approaches, based on integer linear programming and combinatorial branch and bound. Still on the complexity side Joshi et al. (2021) and Joshi et al. (2022) discuss the hardness of a model where customers aim at receiving their food as soon as possible and therefore a tardiness function has to be optimized.

Finally, the model we are interested in is somehow related to the so-called *Same-day Delivery Problem* (Ulmer, Goodson, et al. (2020), Klein and Steinhardt (2022)), in which consumers place orders that will be delivered by a single vehicle or by a fleet of vans (in the same day). In the Same-day Delivery Problem there is 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 capacity of the vehicle(s) to reduce the number of empty trips to the depot. Therefore, this problem is strongly related to the well-known VRP problem with release times (Azi, Gendreau, and Potvin (2012); Archetti, Feillet, and Speranza (2015)).

2 A few data and figures on customers demand

In this section, we provide a few data and figures concerning the customer demand for food delivery in the area of Rome. We restrict to demands collected by $\mathcal{M}$ in the period from January to March 2018, focusing on dinner shifts, as $\mathcal{M}$ receives a larger number of orders in that shift.

Our aim here is twofold: first, we want to motivate some assumptions that are at the base of the optimization models presented later; then, we want to provide a few insights on the food delivery market in the metropolitan area of Rome that may be of interest to other research groups.

We start with Figure 1, that provides the distribution of *placement times*, i.e., the time at which a customer places an order, and *target times*, i.e., the time at which that customer would like to receive that order. Note that, even though Figure 1 deals with orders for the dinner shift, customer might place orders for that shift well in advance, e.g. in the morning (actually even days in advance, but that does not really happen).

Figure 2 and Table 1 build upon Figure 1. Figure 2 plots the distance between the placement time and the target time of a 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 minutes; however, a large fraction of customers prefer to book in advance. Table 1 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 hour in advance* (once again, we here refer to data collected in the period from January to March 2018 for the dinner shift).

We now comment on some data about restaurants. From January to March 2018, 386 restaurants were available to customers through the services of $\mathcal{M}$. However, customers demand was not uniformly distributed among the restaurants: 7 restaurants received 26.5% of orders; 20 restaurant received 40.3% of orders; 29 restaurants received 50.3% of the requests; see Table 2 for more detailed data. It is interesting to cross-reference these numbers and

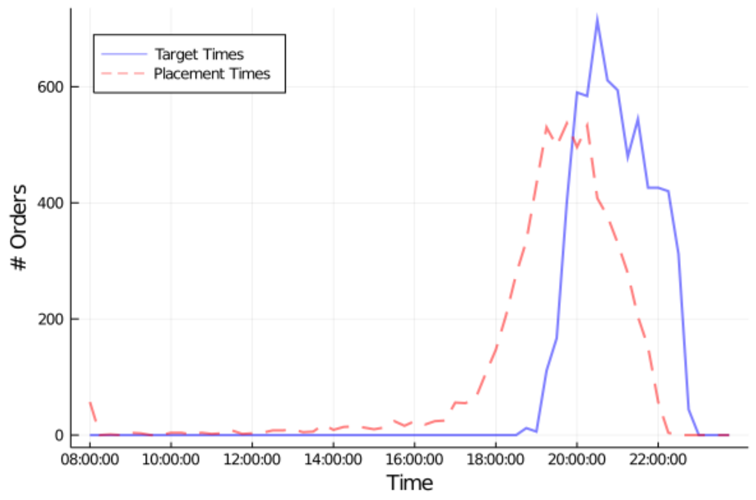

Figure 1: Placement times (red) vs target times (blue) for dinner shifts; note that customers may ask only for deliveries between 7:00 p.m. and 10:30 p.m., however they can place orders largely in advance.

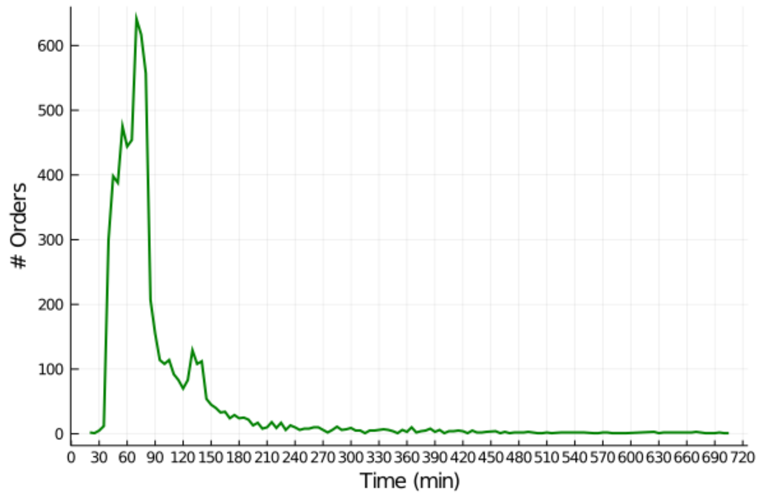

Figure 2: On the x-axis the distance between the placement time and the target time of a same order, on the y-axis the number of orders placed during that shift.

	[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

Table 1: Percentage and cumulative percentage of orders with a target time within n minutes from the placement time (recall that the slack between target time and placement time of a same order cannot be less than 30 minutes).

the *average waiting times* at restaurants, i.e., the average time that couriers have to wait to collect orders at a given restaurant because the order is late with respect to the time agreed with $\mathcal{M}$. As a matter of fact, quite often couriers do indeed *wait* at a restaurant to collect an order, even though restaurant 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 couriers show up, as to avoid that the quality of the food degrades. In Table 2, we report the average waiting time of the 20 most performing restaurants. We point out that 14 restaurants out of 20 have an average waiting time that exceeds the value of 6.76 minutes which is the average value over all restaurants; it seems therefore that in Rome customers are ready to accept a (little) delay when ordering from restaurants they like.

Finally, Figure 3 provides a heat map of the distribution of orders for $\mathcal{M}$ for the city of Rome. It is interesting to evaluate the delivery radius of orders, i.e., the distance between the place where the order is collected (i.e., the location of a restaurant) and the place where the order is delivered (i.e., the address of a customer). According to our data, the average delivery radius of orders is 3.35 km (once again, we here refer to data collected in the period from January to March 2018 for the dinner shift). Note that $\mathcal{M}$ does not impose any restriction on the distance between pickup and delivery so that a customer can order from any restaurant.

2.1 Towards an Optimization Model for Courier Assignment

In this section we start to shape an optimization model for courier assignment. We point out that, while this model is inspired by our collaboration with $\mathcal{M}$, our ambition is nevertheless that of stating a general optimization model that can be used in different settings.

We first discuss the main assumptions behind this model. $\mathcal{M}$ focuses on top-end customers and pursues high Quality of Service through positive commitment of couriers and careful management of delays. Fundamental

Restaurant	Average Waiting Time	% of Orders	Cumulative % of Orders
S_1	8.13	7.78	7.78
S_2	7.00	5.91	13.69
S_3	9.37	3.54	17.23
S_4	7.30	2.81	20.04
S_5	8.28	2.31	22.35
S_6	8.48	2.25	24.60
S_7	6.07	1.91	26.51
S_8	6.37	1.63	28.14
S_9	9.10	1.60	29.74
S_{10}	8.45	1.58	31.33
S_{11}	7.12	1.40	32.72
S_{12}	6.41	1.35	34.07
S_{13}	6.41	1.33	35.41
S_{14}	11.21	1.24	36.65
S_{15}	5.19	1.16	37.81
S_{16}	6.20	1.10	38.92
S_{17}	10.12	1.06	39.97
S_{18}	7.55	1.06	41.03
S_{19}	7.27	1.04	42.07
S_{20}	9.44	0.98	43.05

Table 2: Average waiting times (in minutes) for the 20 most performing restaurants $S_1, \dots, S_{20}$(in terms of number of orders).

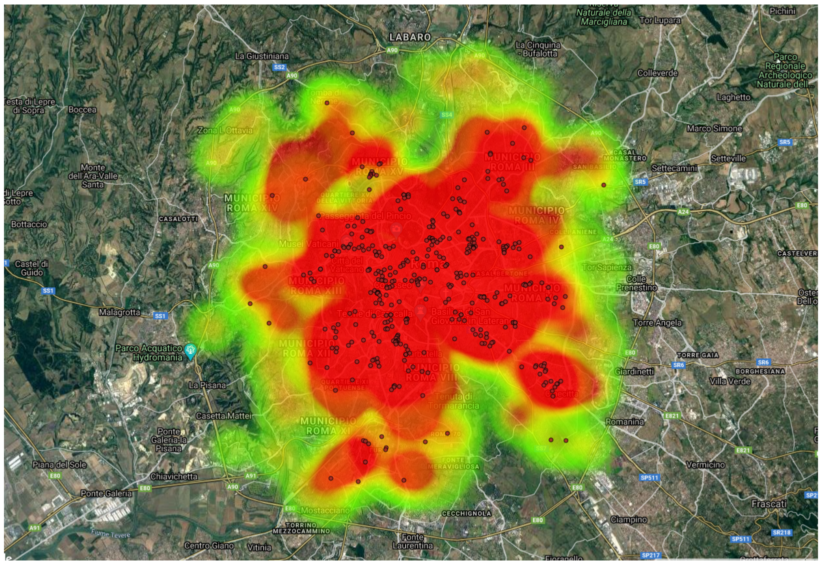

Figure 3: Heat map of the geographic distribution of orders in Rome: dots are restaurants available to customers.

facts about couriers are the followings: (i) couriers are hired before the shift 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 will autonomously choose her route for dispatching an order that has been assigned. As for the management of delays, the key fact is that customers that experience a delay, or even worse a cancellation, 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 cancellations (note that orders with delay larger than 60 minutes will always be cancelled). *Therefore, our optimization model deals with the problem of dispatching orders to couriers as to avoid as much as possible delays and cancellations.*

Another key fact about the optimization model follows from the nature of the demand. As we discussed in Section 2, see Table 1 in particular, only a small fraction of orders are placed on a best effort bases (that is, customers asking for a delivery as soon as possible). Together with $\mathcal{M}$ we therefore decided to develop a solution strategy that is based on the *iterative solution of a fully deterministic problem.* Namely, this strategy builds upon the solution of an optimization problem, that is called the *Offline Couriers Assignment Problem* (OCAP), where we assume a full knowledge on the set of orders to be delivered without making any assumption on their distribution. However, OCAP is cast into a rolling horizon framework, i.e., we build and solve a sequence of instances $I_1, I_2, \dots$ of OCAP, such that I^h inherits from I^{h-1} data and a suitable set of side constraints, while adding the set of orders that have been placed in the interval in between the definition of I^{h-1} and I^h. We will give more details on the rolling horizon approach in Section 4, while in the next section we focus on OCAP.

3 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 k a positive integer, we let $[k]$ denote the set $\{0, 1, 2, \dots, k\}$ (note that we assume that $0 \in [k]$).

Discretization. We discretize time and each shift, i.e., lunch or dinner, can be thought of as a finite set of times $[T]$, that conventionally lasts from time 0 to some time $T \in \mathbb{Z}_+$. In practice, each time step $q \in [T]$ corresponds indeed to time $T_0 + \Delta \cdot q$, where T_0 denotes the start of the shift and Δ is the discretization period.

Restaurants. We are given a set of restaurants S. We assume that all restaurants are open for the entire shift and that, for each $s \in S$, we are given:

a *pickup place* p_s, which is where couriers will collect orders from restaurant s;

$\beta_s \in [T]$, the *average waiting time* for couriers, i.e. the average time that couriers have to wait once arrived at p_s to collect an order from s (see the discussion in Section 2 and in particular Table 2).

Orders. We are given a set of orders O. Each order $i \in O$ has been placed by a customer[3] and we assume that *the customer asks for a delivery from a* single *restaurant from* S. Therefore, for each order $i \in O$, we are given:

a restaurant $s(i) \in S$

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

a *target* time $t'_i \in [T]$ that is the time requested by the customer for the delivery.

In the following, it is convenient to let $p_i := p_{s(i)}$ and $\beta_i := \beta_{s(i)}$ respectively denote the pickup place and the average waiting time of restaurant $s(i)$.

Time Windows and Width. While t'_i is the target time for delivering order i, we however allow for the possibility of delivering i at a time $t \in [T]$ different from t'_i, provided that t belongs to a suitable time-window around t'_i. The time-window is defined by two parameters, $w'_l, w'_r \in \mathbb{N}_+$, so that order i can be indeed delivered at any time in $\{t'_i - w'_l, t'_i - w'_l + 1, \ldots, t'_i + w'_r - 1, t'_i + w'_r\}$. Note that we allow for early deliveries, but the time-window needs not to be symmetric with respect to t'_i and very likely $w'_l << w'_r$. It follows that, for each order $i \in O$, we are also given:

a *time-window* $W_i = \{t_i, t_i + 1, \ldots, t_i + w\}$, where $t_i := t'_i - w'_l$ is the *start time* and $w := w'_l + w'_r$ is the *width*, so that order i can be delivered at any time in W_i (with different costs, though: see the following).

In practice, the width is a parameter of capital importance: the smaller w is, the higher the QoS is since the delivery of i will be more likely at a

[3]For each order there is indeed also a placement time, as it was defined in Section 2, and we will exploit this info in Section 6.3

time close to t'_i. (Note that, in practice, Δ plays a role too because of the discretization.)

Costs. As we discussed in Section 2, we aim at minimizing a suitable cost function that takes into account delays and cancellations, without considering any cost related to the number of couriers (as they are hired before the shift) and to the cost of the routing (as each courier will autonomously choose her route). An order $i \in O$ that cannot be delivered at a time $t \in W_i$ will be cancelled. However, the cancellation of i has a (large) *cancellation cost* $\bar{c}$, as well as there are different costs for delivering i at different times of W_i. Namely, we are given:

a non-decreasing *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 cost functions c is non-decreasing, i.e., is such that $c(h) \leq c(h+1)$ for each $h \in [w-1]$, is quite sensible. 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 much in advance.

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

a *release place* p_j;

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

The courier j will start her shift from place p_j at time r_j. Then *she will be available for deliveries up to time T without any constraint on the location of her last delivery*: possibly at a place far from p_j. As we will later discuss, our integer programming formulation deeply exploits this last assumption (see Section 7).

Travel Times and **Check Out Time.** It follows from above that we can infer a set P of relevant *places*, namely $P := \{p_i, i \in O\} \cup \{d_i, i \in O\} \cup \{p_j, j \in C\}$. We therefore assume that, for each ordered pair of places (x, y), $x, y \in P$, we are given:

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

As discussed in Section 2, a courier that has been assigned an order $i \in O$ will autonomously choose her route for going from pickup place p_i to delivery place d_i. However, we expect the travel time $d(p_i, d_i)$ to be the same for each courier.

Finally, we assume that we are given the following parameter:

$\alpha \in [T]$, the *check-out time*, i.e. the average time for a courier to go from the delivery place (street level) to the customer's door, check out and come back to the street level.

3.1 A Formal Definition of OCAP

We are now ready to provide a formal definition of OCAP. We are given a set of couriers C, a set of restaurants S and so on, as described so far. In particular, we are given a set O of orders and we want to solve the optimization problem of delivering those orders at a minimum cost in an *offline fashion*, i.e., assuming full knowledge of O. 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 cost function c for orders that are delivered and the fixed cost $\bar{c}$ for orders that are canceled.

In order to provide a formal statement of the problem, we need the following definition:

A *feasible schedule* for a courier $j \in C$ is a pair (O_j, σ_j), with $O_j = \{i^j_1, \ldots, i^j_{|O_j|}\} \subseteq O$ and $\sigma_j : O_j \mapsto \{r_j, r_j+1, \ldots, T\}$, such that:

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

(ii) $\sigma_j(i^j_1) \geq r_j + d(p_j, p_{i^j_1}) + \beta_{i^j_1} + d(p_{i^j_1}, d_{i^j_1})$;

(iii) $\sigma_j(i^j_{k+1}) \geq \sigma_j(i^j_k) + \alpha + d(d_{i^j_k}, p_{i^j_{k+1}}) + \beta_{i^j_{k+1}} + d(p_{i^j_{k+1}}, d_{i^j_{k+1}})$, $k = 1..|O_j|-1$.

For each $k = 1..|O_j|$, $\sigma_j(i^j_k)$ is the time at which courier j delivers order i^j_k, and this is done at cost $c(\sigma_j(i^j_k) - t_{i^j_k})$. We therefore let $c(O_j, \sigma_j) := \sum_{k=1..|O_j|} c(\sigma_j(i^j_k) - t_{i^j_k})$.

Hence we aim at finding a collection $(\mathcal{O}, \sigma) = \{(O_1, \sigma_1), \ldots, (O_{|C|}, \sigma_{|C|})\}$ of feasible schedules, one for each courier $j \in C$, such that 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 canceled. Namely, the *Offline Couriers Assignment Problem* is defined as follows:

The Offline Couriers Assignment Problem (OCAP). **Given:** a set S of restaurants; a set O of orders; a set C of couriers; the width w; the cost function $c : [w] \mapsto \mathbb{Z}$ and the cancellation cost $\bar{c}$; travel times d and check out time α; **Find:** a collection $(\mathcal{O}, \sigma) = \{(O_1, \sigma_1), \ldots, (O_{|C|}, \sigma_{|C|})\}$ of feasible schedules, such that the sets O_j are pairwise disjoint and 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 Cosmi, Oriolo, et al. (2019) and Garey and Johnson (1979)).

3.2 The $\mathcal{M}$ Setting

In this section, we tailor the model discussed so far to the $\mathcal{M}$' setting.

Discretization. $\mathcal{M}$ deals with two shifts, lunch and dinner, that respectively last from 11:30 p.m. to 2:30 p.m. and from 6:30 p.m. to 10:30 p.m. However, customers may ask for deliveries only between 12:00 p.m. and 2:30 p.m. and between 7:00 p.m. and 10:30 p.m., as the first 30 minutes of each shift are only "available" to couriers to reach the location of a pickup from their release places. In both cases, we set the discretization period Δ equal to 5 minutes. Therefore, according to our definitions, the set of times for lunch is $[T]$, with $T = 30$, and the set of times for dinner shift is $[T]$, with $T = 42$.

Orders. While customers may ask for deliveries between 12:00 p.m. and 2:30 p.m. (lunch) and between 7:00 p.m. and 10:30 p.m. (dinner), only a few times in those intervals can be indeed selected for a delivery, namely those at :00, :15, :30, :45. It follows that at lunch (respectively, dinner) there are 11 (respectively, 15) possible values for the target time t_i' of an order (and of course the same applies to the start time t_i).

Width. In the $\mathcal{M}$ model, $w_l' = 3$ and $w_r' = 12$, so that the width $w := w_l' + w_r'$ is equal to 16. Therefore the time window W_i for order i is $\{t_i, t_i{+}1, \ldots, t_i{+}16\}$. In other words, if we "reverse" the discretization and the normalization, we allow for an order i to be delivered at any time in the interval $[\theta_i - 15, \theta_i + 60]$, where $\theta_i = T_0 + 5 \cdot t_i + 15$ is the (non-normalized) target time selected by the customer (T_0 is equal to 11:30 a.m for lunch and to 6:30 p.m for dinner).

Costs. The cost $\bar{c}$ of a cancellation is set equal to 10, while the non-decreasing cost-function $c : [w] \mapsto \mathbb{Z}$ is defined as follows:

- $c(h) = 0$ for $h \in \{0, 1, \ldots, 6\}$;
- $c(h) = 1$ 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 computed from geographical coordinates using standard packages. The checkout time α is set equal to 1 time step, i.e., 5 minutes.

4 A Rolling Horizon Approach

The Offline Couriers Assignment Problem assumes a complete knowledge of the set of orders to be delivered. While this setting is consistent with the fact that a large fraction of the orders are placed early in advance (see the discussion in Section 2), there are still orders that are placed dynamically during the shift (very likely on a best effort base) and therefore are not known in advance. In the following, we show how to cast OCAP into a rolling horizon strategy, as to deal with a dynamic setting, where orders are placed now and then. The idea is that of solving a sequence of instances of OCAP $I^1, I^2, \ldots$, each with a creation time $t^1, t^2, \ldots$, such that I^h inherits from I^{h-1} data and a suitable set of side constraints, while adding the set of orders that have been placed (by customers) in the interval $(t^{h-1}, t^h]$.

We need a few definitions. Recall (see Section 3.1) that an instance $I = (O, S, C, w, c, \bar{c}, \alpha, d)$ of OCAP is defined by: a set S of restaurants; a set O of orders; a set C of couriers; the width w; the cost function $c : [w] \mapsto \mathbb{Z}$; the cancellation cost $\bar{c}$; travel times d; checkout time α. Recall also that a solution to the instance I is a collection $(\mathcal{O}, \sigma) = \{(O_1, \sigma_1), \ldots, (O_{|C|}, \sigma_{|C|})\}$ of feasible schedules, such that the sets O_j are pairwise disjoint. Moreover, a *feasible schedule* (O_j, σ_j), with $O_j = \{i^j_1, \ldots, i^j_{|O_j|}\}$ and $\sigma_j : O_j \mapsto \{r_j, r_j + 1, \ldots, T\}$, satisfies:

(ii) $\sigma_j(i^j_1) \geq r_j + d(p_j, p_{i^j_1}) + \beta_{i^j_1} + d(p_{i^j_1}, d_{i^j_1})$;

We are interested in particular in the first order i_1^j that is assigned by the feasible schedule to courier j (if any): we say that order i_1^j is *incumbent* for j. We are also interested in the last time by which courier j must leave her release place r_j in order to deliver her incumbent order i_1^j at time $\sigma_j(i_1^j)$: we denote this time by either x_j (if we want to focus on the courier) or $x_{i_1^j}$ (if we want to focus on the order) and we call it the *threshold time* for both courier j and order i_1^j: namely, $x_j := x_{i_1^j} := \sigma_j(i_1^j) - (d(p_j, p_{i_1^j}) + \beta_{i_1^j} + d(p_{i_1^j}, d_{i_1^j}))$. The rationale behind this definition is that, if we want to cast solution $(\mathcal{O}, \sigma)$ into a dynamic setting, we can change the assignment of (incumbent) order i_1^j till time $x_{i_1^j}$, and analogously we can change the schedule of courier j till time x_j.

We now show by induction how to build a sequence of instances $\{I^h, h = 1..m\}$ of OCAP. Each instance I^h will have a *creation time* $t^h = t^0 + (h-1)\gamma$, for some suitable t^0 and γ, and will therefore "last" for time γ: we indeed set $\gamma := \alpha$ so that in the time covered by each instance no courier can deliver more than one order. All instances $\{I^h, h = 1..m\}$ will share the same values for S, w, c, $\bar{c}$, d and α. By induction assume that we are given an instance I^{h-1} and a feasible (possibly optimal) schedule $(\mathcal{O}^{h-1}, \sigma^{h-1})$. We build I^h from I^{h-1} by updating: 1) the set of orders, as to include the orders that have been placed in the interval $(t^{h-1}, t^h]$ and remove incumbent orders whose threshold time was within t^{h+1}; 2) the release place and release time of each courier, again on the base of their threshold time. Namely:

Base Step. Definition of I^1: The set of orders, that we denote as $O(I^1)$, is made of all orders that have been placed by time t^1. For each courier $j \in C$, release place p_j and release time r_j are those that have been communicated at the beginning of the shift. We compute a feasible schedule $(\mathcal{O}^1, \sigma^1)$ and then infer threshold time x_i^1 for each order that is incumbent for some courier, and threshold time x_j^1 for each courier $j \in C$ (set the threshold time $x_j^1 := T$ if no order is assigned to j by $(\mathcal{O}^1, \sigma^1)$).

Induction Step. Building I^h from I^{h-1}: For $h \geq 2$, we are given the instance I^{h-1} and a feasible schedule $(\mathcal{O}^{h-1}, \sigma^{h-1})$, with threshold x_i^{h-1} for each order i that is incumbent for some courier, and threshold x_j^{h-1} for each courier $j \in C$. Then define I^h as follows:

the set of orders in I^h is made of all orders that have been placed in the interval $(t^{h-1}, t^h]$ and all orders in I^{h-1}, but for orders i that, according to the current schedule $(\mathcal{O}^{h-1}, \sigma^{h-1})$, have to be served within the current time window $(t^h, t^{h+1}]$ (i.e. those with threshold time $x_i^{h-1} \leq t^{h+1}$);

for each courier $j \in C$, let i_1^j be the order that is incumbent for j (if any) and let x_j^{h-1} be the threshold time for j. Then set the release place and time of j respectively as $d(i_1^j)$ and $\sigma_j^{h-1}(i_1^j) + \alpha$ (once again note that j cannot deliver more than one order in the period covered by I^h, since $\gamma := \alpha$).

Compute a feasible (possibly optimal) schedule $(\mathcal{O}^h, \sigma^h)$ and then infer threshold time x_i^h for each order i that is incumbent for some courier and threshold time x_j^h for each courier $j \in C$ (set $x_j^h := x_{j-1}^h$, if no order of I^h is assigned to j).

5 An Integer Linear Programming Formulation for OCAP

In this section we provide an Integer Linear Programming formulation for OCAP. The formulation builds upon a suitable time indexed network that we present later in this section. One key issue for keeping the size of this network under control is focusing on a particular class of feasible schedules, that, in accordance with the scheduling literature, we call early start.

5.1 Early start schedules

Note that, in general, for a given and ordered subset of orders $O_j = \{i_1^j, \ldots, i_{|O_j|}^j\}$ assigned to a courier $j \in C$, there might exist different functions $\sigma_j', \sigma_j'' : O_j \mapsto \{r_j, r_j + 1, \ldots, T\}$ such that both (O_j, σ_j') and (O_j, σ_j'') are feasible schedules. However, there might be only one function σ_j such that the pair (O_j, σ_j) is a feasible *early start* schedule, as in an early start schedule the function 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 $i_1^j, \ldots, i_{|O_j|}^j$. Namely, (O_j, σ_j) is a (feasible) early start schedule if the followings hold:

(i) $\sigma_j(i_k^j) \in \{t_{i_k^j}, t_{i_k^j} + 1, \ldots, t_{i_k^j} + w\}$, for each $k = 1..|O_j|$;

(ii) $\sigma_j(i_1^j) = \max\{t_{i_1^j}, r_j + d(p_j, p_{i_1^j}) + \beta_{i_1^j} + d(p_{i_1^j}, d_{i_1^j})\}$;

(iii) $\sigma_j(i_{k+1}^j) = \max\{t_{i_{k+1}^j}, \sigma_j(i_k^j) + \alpha + d(d_{i_k^j}, p_{i_{k+1}^j}) + \beta_{i_{k+1}^j} + d(p_{i_{k+1}^j}, d_{i_{k+1}^j})\}$, for each $k = 1..|O_j| - 1$.

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

Lemma 1. *If the cost function c is non-decreasing, then there exists an optimal solution to* OCAP *where, for each courier, the schedule is early start.*

5.2 A Time-Expanded Network

We are given an instance $(O, S, C, w, c, \bar{c}, \alpha, d)$ of OCAP, i.e., a set S of restaurants, a set O of orders, a set C of couriers, the width w, the cost function $c : [w] \mapsto \mathbb{Z}$, the cancellation cost $\bar{c}$, travel times d and checkout time α. In the following, we show how to associate to this instance a suitable time indexed network $Q(N, A)$. We also illustrate the main features of our construction by means of a simple example.

Example 2. *We consider the following instance: we are given a set $S = \{s_1, s_2\}$ of restaurants with $\beta_{s_1} = \beta_{s_2} = 1$, a set $O = \{1, 2\}$ of orders such that order i is from restaurant s_i, a single courier j, width $w = 15$, travel times such that $d(p_j, p_{s_1}) + d(p_{s_1}, d_1) = 11$, $d(p_j, p_{s_2}) + d(p_{s_2}, d_2) = 9$, $d(d_1, p_{s_2}) + d(p_{s_2}, d_2) = 12$, $d(d_2, p_{s_1}) + d(p_{s_1}, d_1) = 1$, target times $t_1 = 4$, $t_2 = 10$, and $\alpha = 1$. Here, for our purposes, costs are non-relevant, so we skip them.*

We start with defining the set of nodes N. It is convenient to associate with each courier $j \in C$ a *layer*, even though the layer is made of a single node $(j, 0)$. The node $(j, 0)$ stands for j being ready at her release place at time r_j.

To the contrary, the *layer* of an order $i \in O$ is made of $w+1$ nodes (recall that w is the width). Namely, we define the nodes (i, h) for each $h \in [w]$. The node (i, h) stands for the delivery of order i at time $t_i + h$.

We now define the set of arcs A. Each arc has a cost, which is possibly 0. A first set of arcs, that we call *horizontal*, are defined in the layer of orders. 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 place of order i after the check out.

A second set of arcs runs from couriers layer to orders layer. Consider therefore a courier $j \in C$ and an order $i \in O$. An arc from $(j, 0)$ to the layer of i stands for i being the first order delivered in the shift of j. Let $\delta_{j,i} := r_j + d(p_j, p_i) + d(p_i, d_i) + \beta_i$, that is the earliest time j can serve i. For the sake of clarity, we delve into three cases:

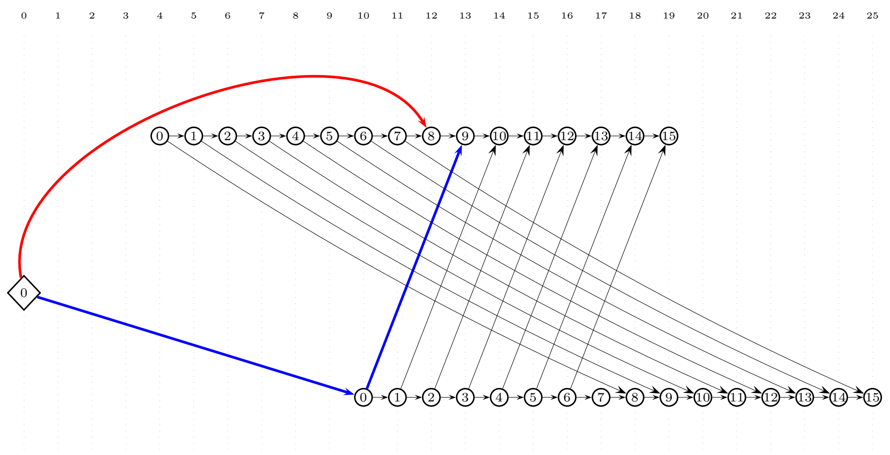

Figure 4: See Example 2. The node layer of order 1 (resp. 2) is above (resp. below) and order nodes are represented by circles labelled with h. The courier j is associated with the 0 diamond node. Recall that $\beta_{s_1} = \beta_{s_2} = 1$, $w = 15$, $d(p_J, p_{s_1}) + d(p_{s_1}, d_1) = 11$, $d(p_J, p_{s_2}) + d(p_{s_2}, d_2) = 9$, $d(d_1, p_{s_2}) + d(p_{s_2}, d_2) = 12$, $d(d_2, p_{s_1}) + d(p_{s_1}, d_1) = 1$, $t_1 = 4$, $t_2 = 10$, and $\alpha = 1$. Note that there exists a feasible schedule that serves order 2 and then order 1, but no one exists that swaps the orders.

[$\delta_{j,i} > t_i + w$] In this case, even leaving from p_j at the release time r_j, the courier cannot reach d_i by time $t_i + w$, which is the last time for delivering i: therefore there are no arcs from layer j to layer i;

[$t_i + w \geq \delta_{j,i} \geq t_i$] In this case, if the courier leaves p_j at the release time r_j, she will reach d_i within the time window $[t_i, t_i + w]$: therefore there will be a single arc from layer j to layer i, and it will be the arc $((j, 0), (i, \delta_{j,i}))$;

[$t_i > \delta_{j,i}$] In this case, if the courier leaves p_j at the release time r_j, then she will anyhow reach d_i before time t_i.There is therefore an *idle time* of size $t_i - \delta_{j,i}$ that the courier can spend either at the release place p_j, or at the pickup place p_i or fractionally at both places . . . however that will not affect the final cost of the solution and so we simply put a single arc from layer j to layer i, namely the arc $((j, 0), (i, 0))$.

The last set of arcs runs from an order layer to another order layer. Consider therefore orders i and l in O. An arc going from the layer of i to the layer of l stands for l being delivered right after i (by a same courier).

Let $\delta_{i,l} := t_i + \alpha + d(d_i, p_l) + \beta_l + d(p_l, d_l)$ to be the earliest time j can deliver l after served i at time t_i. We delve into four cases:

[$\delta_{i,l} > t_l + w$] In this case, even leaving from d_i at time $t_i + \alpha$, which is the first time to checkout order i, it is not possible to reach d_l by time $t_l + w$, which is the last time for delivering l: therefore there are no arcs from layer i to layer l;

[$t_l + w \geq \delta_{i,l} \geq t_l$] In this case, if the courier leaves from d_i at time $t_i + \alpha$, she will reach d_l within the time window $[t_l, t_l + w]$. Therefore, the arcs going from layer i to layer l are the following: $((i, 0), (l, \delta_{i,l})), ((i, 1), (l, \delta_{i,l} + 1)), ..., ((i, w - \delta_{i,l}), (l, w))$.

[$t_l > \delta_{i,l} \geq t_l - w$] In this case, $t_i + \alpha + t_l - \delta_{i,l}$ is the last time for a courier to depart from d_i and still be able to reach d_l by time t_l. The arcs going from layer i to layer l are the following: $((i, t_l - \delta_{i,l}), (l, 0)), ((i, t_l - \delta_{i,l} + 1), (l, 1)), ..., ((i, w), (l, w + t_l - \delta_{i,l}))$. We point out that in this case there are no arcs that are going from nodes $(i, 0), \ldots, (i, t_l - \delta_{i,l} - 1)$ to layer l: however, in order to go e.g. from node $(i, 0)$ to layer l there is the path $(i, 0), (i, 1), \ldots, (i, t_l - \delta_{i,l}), (l, 0))$.

[$t_l - w > \delta_{i,l}$] In this case, even leaving from d_i at time $t_i + \alpha + w$, which is the last time to checkout order i, the courier will reach d_l before time t_l, which is the first time for delivering l. This case is indeed similar to case [$\delta_{j,i} \leq t_i$] discussed above, and we again put a single arc from layer i to layer l, namely the arc $((i, w), (l, 0))$. It follows that there are no arcs going from nodes $(i, 0), \ldots, (i, w - 1)$ to layer l, however, as for case [$t_l - w \leq \delta_{i,l} \leq t_l$], there are paths.

We now deal with arc costs. Note that each arc is entering into a node (i, h), for suitable $i \in O$ and $h \in [w]$. We give each arc a entering into 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.

Consider now a courier $j \in C$ and a feasible early start schedule (O_j, σ_j), with $O_j = \{i^j_1, \ldots, i^j_{|O_j|}\} \subseteq O$ and $\sigma_j : O_j \mapsto [r_j, T]$. Let $s := |O_j|$. It is straightforward to associate to (O_j, σ_j) the following path P_j in $Q(N, A)$:

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

with $b_k \in [w - \sigma_j(i^j_k)]$, for $k = 1..s$, being the idle time of j in d_{p_k}. Note that, by construction, the cost $c(P_j)$ of this path, which is the sum of the costs of its arcs, is equal to $c(O_j, \sigma_j)$. Note also that the path P_j might visit several nodes in the layer corresponding to order i, i.e. $b_i > 0$: namely, that happens if the courier is idle, waiting for the first possible pickup time for order $i+1$. Moreover P_j visits each node layer at most once, i.e., it will not come back to a node layer that has been already visited.

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 $h \in [w]$. Then derive P'_j by shortening P_j until either $h = 0$ or $(i, h-1) \not{i}n P_j$ and by short-cutting P_j (if it visits the same node layer more than once. Note that, by construction, the sequence of nodes visited by P'_j is:

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

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

If we let $O_j := \{i_1, \ldots, i_s\}$ and $\sigma_j(i_k) := t_{i_k} + h_k$, for $k = 1..s$, then the schedule (O_j, σ_j) is feasible and moreover $c(O_j, \sigma_j) = c(P'_j) \leq c(P_j)$.

It follows from above that each feasible early start schedules (O_j, σ_j) correspond to a path P_j in $Q(N, A)$ starting from O_j, and vice versa every path P_j in $Q(N, A)$ starting from O_j is dominated by another one (possibly the same P_j) that corresponds to a feasible schedule. Moreover this correspondence "preserves" costs.

5.3 An Integer Linear Programming Formulation

Following the discussion in the previous section, we may reduce OCAP to the problem of finding a collection of paths $\mathcal{P}$ in $Q(N, A)$, such that:

- each path in $\mathcal{P}$ leaves from a node $(j, 0)$ associated with a courier $j \in C$, and for each courier $j \in C$ there is at most one path leaving from node $(j, 0)$;
- the cost $c(\mathcal{P})$ is minimized. The cost $c(\mathcal{P})$ is made of two terms: the sum of the costs of the arcs in the paths; the cost of cancellations, namely, $\bar{c}$ times the number of orders that have not been assigned to any courier, i.e., $\bar{c}$ times the number of node layers not visited by any path in $\mathcal{P}$.

It is then almost straightforward to provide an integer linear programming formulation for OCAP. However, we spend a few words on the latter term of the cost function, i.e., the cost of cancellations. An order $i \in O$ is delivered if there is a path $P \in \mathcal{P}$ that visits the node layer of i, and therefore if some arc "entering" the layer of i belongs to P. We therefore denote by $\delta^-(i)$ the set of arcs entering the layer of i (i.e., the arcs entering the cut induced by the set of nodes $\bigcup_{h\in[w]}(i,h)$):

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

where as usual we respectively denote by $\delta^-(v)$ and $\delta^+(v)$ the set of arcs entering and leaving a node v. We may therefore formulate OCAP as follows:

$$\begin{aligned}
&\min \sum_{i\in O}(\bar{c}\cdot w_i + \sum_{h\in[w]}\sum_{a\in\delta^-(i,h)} c_a \cdot y_a) \\
&\sum_{a\in\delta^-(i)} y_a + w_i = 1 \qquad i \in O && \text{(i)}\\
&\sum_{a\in\delta^+((i,h))} y_a \leq \sum_{a\in\delta^-((i,h))} y_a \qquad i\in O, h\in[w] \\
&\sum_{a\in\delta^+((j,0))} y_a \leq 1 \qquad j \in C \\
&y \in \{0,1\}^{|A|}, w \in \{0,1\}^{|O|}
\end{aligned}$$

where w_i will be equal to 1 if and only if the order i is canceled (note however that w_i may be easily projected away).

We point out that the formulation above is a "flow-like" formulation, but for the constraints (i). In other words, if we skip constraints (i), then the constraints matrix is a essentially a network matrix and therefore totally unimodular. In the following, we will then refer to this formulation as *F-formulation.*

We will later show, see Sections 6.2 and 6.5.1, that the F-formulation is quite effective. We believe that is due to several (correlated) facts: first, it has good quality in terms of integrality gap; then, by restricting to early start schedules, we are able to keep under control the size of the programs; finally, such a (single commodity) formulation exploits the fact that there are no constraints on the end of the shift for a courier. As soon as we impose constraints on that, namely a given courier has to quit at a different time than T or at time T she has to be back at her release place, then we are not able to extend the above formulation as we need to use multi commodity flow variables.

6 Computational Tests

In this section, we provide some computational results with the aim of validating both the use of the F-formulation to solve OCAP and the rolling horizon approach presented in Section 4 to deal with the dynamic setting.

Once again, we point out that OCAP assumes that all orders are known in advance, without taking into account the placement time of an order (placement times were defined in Section 2). However, the setting of OCAP is such that, for a fixed set of orders, its optimal solution provides a lower bound to any dynamic strategy that deals with the same set of orders while taking into account their placement times.

All the computational experiments have been carried out on a machine with a CPU Intel Xeon E5620 at 2.4GHz and with 48 GB RAM and made use the MIP solver of Gurobi 9.02. We also designed an heuristic procedure so to provide an incumbent solution to the solver. Our tests run on a set of instances derived from real data collected by $\mathcal{M}$. We start describing such instances in the offline setting of OCAP.

6.1 OCAP Instances

We focus on the same data analyzed in Section 2 and derive 33 instance $I_1, \ldots, I_{33}$ of OCAP. We give more details in the following. Recall that, in order to define an instance of OCAP, we need to define: a set S of restaurants, a set O of orders, a set C of couriers, the width w, the cost function $c : [w] \mapsto \mathbb{Z}$, the cancellation cost $\bar{c}$, travel times d and checkout time α. As for $w, c, \bar{c}, d$ and α we stick to the values described in Section 3.2 (we use the package Geodesics.jl for computing geographical coordinates). In order to define an instance we are therefore left with defining the triple (S, O, C).

We consider 33 different triples (S, O, C) leading to the 33 instance $I_1, \ldots, I_{33}$. Each triple was inferred by the data set of a dinner shift for a day in between January and March 2018. Namely, we consider the entire set of orders that were indeed placed on that night, with the same data for each order (target time, restaurant etc.), and the same set of couriers that were hired for that shift, again with the same data (release place etc.). As a result we got a set of 33 instances where the number of orders vary from 169 to 239; the number of couriers from 26 to 50; the ratio between the number of orders and the number of couriers from 4.14 to 6.88: in the following we refer to this value simply as the *ratio*. A few details about these instances are given in the first seven columns of Table 3.

Instance	Ratio	Orders	Couriers	β (mins)	$d_{pick,del}$ (mins)	$d_{del,pick}$ (mins)	Opt(Heu)	Timeopt (secs)
$I1$	6.88	179	26	7.05	6.73	3.26	0	0.03
$I2$	6.64	239	36	6.8	6.19	3.09	0	0.10
$I3$	6.57	197	30	7.1	7.66	3.2	0	0.05
$I4$	6.48	188	29	6.9	6.44	3.31	0	0.05
$I5$	6.37	172	27	7.35	7.3	3.09	0(1)	112.46
$I6$	6.26	213	34	7.2	6.74	2.91	0	0.09
$I7$	6.24	237	38	7.05	5.51	3.06	0	0.13
$I8$	6.2	186	30	7.1	6.88	3.1	0	0.05
$I9$	6.15	209	34	7.3	5.65	2.99	0	0.05
$I10$	6.06	212	35	6.95	6.82	3.08	0(2)	57.93
$I11$	6.06	200	33	7	7.55	3.24	0	0.06
$I12$	5.93	178	30	6.7	6.97	3.11	0	0.05
$I13$	5.89	212	36	6.8	7.34	3.11	0	0.06
$I14$	5.87	182	31	7.25	6.45	2.98	0	0.05
$I15$	5.83	169	29	6.85	6.84	3.25	0	0.05
$I16$	5.79	197	34	7.1	6.4	3.12	0	0.05
$I17$	5.77	202	35	7.35	7.01	3.14	0	0.06
$I18$	5.68	176	31	7.1	6.99	3.07	0	0.05
$I19$	5.58	173	31	7.05	6.45	2.92	0	0.05
$I20$	5.56	217	39	7.4	6.43	3.13	0	0.09
$I21$	5.55	183	33	7.3	6.91	3.29	0	0.05
$I22$	5.5	187	34	7.1	6.56	3.16	0	0.05
$I23$	5.38	172	32	7.25	6.98	3.17	0	0.04
$I24$	5.35	182	34	6.85	7.31	3.04	0	0.05
$I25$	5.34	219	41	6.75	6.53	3.19	0	0.03
$I26$	5.26	184	35	7	6.49	3.15	0	0.05
$I27$	5.19	187	36	6.85	6.36	2.87	0	0.04
$I28$	4.91	221	45	6.85	6.76	3.26	0	0.10
$I29$	4.9	191	39	7.2	7.15	3.01	0	0.06
$I30$	4.86	180	37	7.4	6.31	3.04	0	0.04
$I31$	4.73	194	41	6.75	6.13	3.07	0	0.05
$I32$	4.48	197	44	7.15	6.17	2.99	0	0.09
$I33$	4.14	207	50	7.05	7.01	3.12	0	0.07

Table 3: This table refers to 33 real-world instances provided by $\mathcal{M}$. For each instance, the table reports: the ratio between the number of orders and the number of couriers; the number of orders; the number of couriers; the average waiting time β; the average pickup to delivery time; the average delivery to pickup; the value of the optimal solution (value of the heuristic solution, if different); the elapsed time of heuristic and solver.

6.2 Computational results for OCAP Instances

As a matter of fact, these instances turned out to be quite "easy" (see the last columns of Table 3): in 31 out of 33 cases the heuristic solution could be computed in at most 0.13 secs, had value 0 and was therefore optimal. In the other two cases, the incumbent heuristic could be computed in at most 0.24 secs, and had value respectively 1 and 2: Gurobi was launched and optimal solutions, again with value 0, could be found in respectively 57.93 and 112.46 secs.

We believe that these results (together with those provided in Section 6.5.1) show the good quality of the F-formulation. Nevertheless one may wonder why these instances turned up to be so easy. First, we point out that at the time when these instances were collected $\mathcal{M}$ was using a manual dispatching of the orders that, as a matter of fact, did not perform very well: therefore, in order to reduce delays (and discount vouchers to compensate them, see Section 2.1), they were hiring too many couriers. Then, there is also a catch: in solving OCAP we assume that all orders are known in advance, and this likely makes it easier to find good solutions. In the next section we will therefore cast these instances into a scenario where orders arrive dynamically. We will anyhow go back to the problem of estimating the right number of couriers for a shift in Section 6.5.

6.3 Casting OCAP instances into a dynamic setting

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 4 and test the rolling horizon approach. Therefore, for each instance $I_i, i = 1..33$, we build a sequence of sub-instances $\{I_i^h, h = 1..p\}$, each with a creation time $t^h = (h-1)\gamma$. We set $\gamma := 1$, i.e., we set the time distance between two sub-instances equal to one time step and therefore, following Section 3.2, equal to 5 minutes. 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 in I_i with placement time in the interval $(t^{h-1}, t^h]$.

6.4 Computational results for the dynamic setting

For $i = 1..33$, we solved the sequence of sub-instances $\{I_i^h, h = 1..p\}$ making again use of the MIP solver of Gurobi 9.02 (and an heuristic to provide an incumbent to the solver). However, for each instance I_i^h, we set a time limit of 5 minutes for the overall computation, i.e., we stop the computation if

Offline Opt	Rolling Heur	Rolling Solver	Opt Solver
0.00	0.76	0.03	91.67%

Table 4: Testing the rolling horizon approach. For instances $I_1, \ldots, I_{33}$, we report: the average value of the solutions to OCAP found by the solver in the offline setting (Offline Opt); the average value of the solutions found when using the heuristic in the rolling horizon approach (Rolling Heur); the average value of the solutions found when using both the heuristic and the solver in the rolling horizon approach (Rolling Solver); in the latter case, the average percentage of times the solver was able to find an optimal solution within the time limit of 5 minutes - when it was called (Opt Solver).

it is still running when reaching the next time step (in these cases, as the heuristic is very quick, we indeed stop the MIP solver). 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 are shown in Table 4. We point out that there was only one instance where the schedule found by the rolling horizon approach caused a delay, and it was a delay within 15 min. Although these results suggest that the rolling horizon approach is suitable to deal with a dynamic setting, it might be simply the case that $I_1, \ldots, I_{33}$ are "easy" also when cast into the dynamic setting, just because too many couriers were hired for the corresponding shifts (as we discussed in Section 6.2). In order to provide more challenging and meaningful computational tests, we therefore derived from $I_1, \ldots, I_{33}$ other instances by simply reducing the number of couriers that are available for each instance (shift). We report on these experiments in the next section.

6.5 Reducing the number of couriers

The results in Table 4 suggest that the number of couriers could be reduced in each instance without introducing delays (or too many delays). It is indeed easy to slightly change the F-formulation so as to compute, for each instance of OCAP, the minimum size of a set of couriers such that orders can be delivered without any delay (and of course without any cancellation) in the offline setting. We ran this experiment for instances $I_1, \ldots, I_{33}$ and found out that, for each instance, a number of couriers that is at least $\frac{1}{7.14}$ of the number of orders is enough to find a feasible schedule without delays.

Ratio	Heu	Opt	Late_1	Late_2	Late_3	Canc	Root Gap	Nodes	Time (secs)
7.0	4.9	0.0	0.0	0.0	0.0	0.0	0.0	0.2	95.8
7.5	15.2	0.2	0.2	0.0	0.0	0.0	1.4	333.5	635.2
8	33.5	2.6	1.0	0.5	0.2	0.0	6.7	476.0	661.6
8.5	60.8	9.3	3.8	1.5	0.8	0.0	8.7	6210.9	2852.2
9	94.8	23.5	7.9	2.7	3.0	0.1	12.1	18776.8	8087.8

Table 5: Solving OCAP model for high ratios. Instances are grouped by ratio value. For each group, we report: the ratio value, the average value of the heuristic solution (Heu), the average value of the optimal solution (Opt), the average number Late_1 of orders delivered with a delay within $[15, 30)$ mins, the average number Late_2 of orders delivered with a delay within $[30, 45)$ mins, the average number Late_3 of orders delivered with a delay within $[45, 60)$ mins, the average number Canc of cancelled orders (because the delay in the delivery time would be larger than 60 mins), the average percentage gap at the root node (Root gap), the average number of branching nodes explored by the MIP solver (Nodes), and the average total computing time.

Following the discussion from previous section, we therefore derived more challenging instances from $I_1, \ldots, I_{33}$ by randomly removing some couriers as to impose a ratio between the number of orders and the number of couriers that is 7 or more. Namely we set the ratio equal to 7, 7.5, 8, 8.5, and 9, and therefore derived 165 instances of OCAP.

6.5.1 Reducing the number of couriers in the offline model

We solved the 165 instances of OCAP within the same framework presented in Section 6.2. Results are reported in Table 5. As one would expect, when the ratio grows, the computation becomes more challenging and the cost of the optimal solution increases. Nevertheless, on the one side, all instances could be solved within 3 hours of computational time. We think this is due to the good quality of the F-formulation, as certified by the good lower bounds achieved at the root node (the average percentage gap at root is, at most, 12.1%). It is also worthwhile mentioning once again that dealing with early start schedules (see Section 5.1), allows to keep under control the size of the integer linear programs and therefore computational times.

Ratio	Offline Opt	Rolling Heur	Rolling Solver	Opt Solver
7	0.0	17.8	2.1	89.6%
7.5	0.2	36.3	7.8	92.9%
8	2.6	61.6	17.6	94.2%
8.5	9.3	82.8	35.0	92.9%
9	23.5	105.8	57.6	92.6%

Table 6: Testing the rolling horizon approach for high ratios. For each ratio in range 7-9, we report: the average value of the solutions found in the offline setting (Offline Opt); the average value of the solutions found when using the heuristic in the rolling horizon approach (Rolling Heur); the average value of the solutions found when using both the heuristic and the solver in the rolling horizon approach (Rolling Solver); in the latter case, the average percentage of times the solver was able to find an optimal solution within the time limit of 5 minutes(Opt Solver).

6.5.2 Reducing the number of couriers in the dynamic setting

We solved the 165 instances of OCAP by the rolling horizon approach. Results are reported in Table 6.

We extract a few interesting facts from Table 6. First, when the ratio increases, the use of the solver (and of the F-formulation) becomes critical in order to get good solutions, and interestingly the solver is able to find an optimal solution within one time step in more than 90% of the calls. Then, the results show that the rolling horizon approach is indeed suitable even when the ratio (reasonably) increases; in order to give a better picture of this behaviour, we plot in Figure 5 the average difference between the value of the optimal offline solution to OCAP and the value of the solution found by the rolling horizon approach (note that we evaluated these values also for small ratios).

7 Concluding remarks

We investigated 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 as to avoid as much as possible delays and cancellations. At the core of these models there is the solution through integer programming of a fully deterministic optimization problems, the Offline

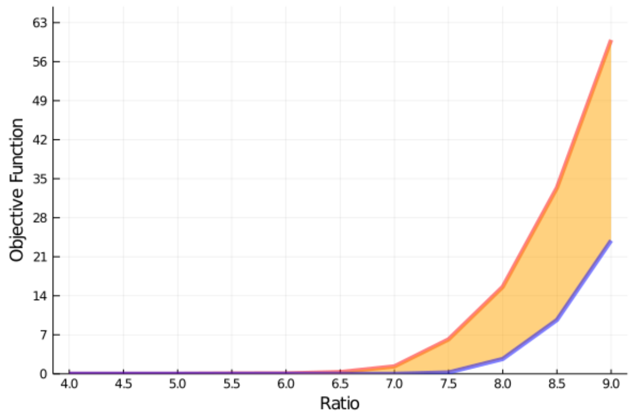

Figure 5: The blue and red lines represent respectively the average values of the offline solutions to OCAP and the average values of the solutions found by the rolling horizon approach, for ratios ranging from 4.5 to 9.

Couriers Assignment Problem, and, in particular, a suitable “flow-like” formulation for the latter. We validated both the models and the F-formulation through some computational tests on some real instances that showed their effectiveness.

We point out that our models can easily handle the case where we assume in that problem that each order has its own width and its own cancellation cost, which in practice is a simple way for providing higher QoS to valuable customers (e.g. the width could be smaller and the cancellation cost higher for their orders). Also our models would work with non-regular cost function, but in this case we expect to pay some computational cost. Finally, a natural extension of the Offline Couriers Assignment Problem would be that of considering orders aggregation (e.g. a courier could pick different orders from a same restaurant), but this was not considered in this research because it partially conflicts with the maximization of QoS which was the main focus of $\mathcal{M}$.

References

Agnetis, Alessandro, Matteo Cosmi, Gaia Nicosia, and Andrea Pacifici (2021). “An order aggregation and scheduling problem for meal delivery”. In: *Optimization-Online*. URL: `http://www.optimization-online.org/DB_HTML/2021/05/8392.html`.

Archetti, Claudia, Dominique Feillet, and M. Grazia Speranza (2015). "Complexity of routing problems with release dates". In: *European Journal of Operational Research* 247(3), pp. 797–803. ISSN: 0377-2217.

Azi, Nabila, Michel Gendreau, and Jean-Yves Potvin (2012). "A dynamic vehicle routing problem with multiple delivery routes". In: *Annals of Operations Research* 199(1), pp. 103–112.

Böhm, Martin, Nicole Megow, and Jens Schlöter (2021). "Throughput Scheduling with Equal Additive Laxity". In: *Algorithms and Complexity*. Ed. by Tiziana Calamoneri and Federico Corò. Springer International Publishing, pp. 130–143.

Bozanta, Aysun, Mucahit Cevik, Can Kavaklioglu, Eray M. Kavuk, Ayse Tosun, Sibel B. Sonuc, Alper Duranel, and Ayse Basar (2022). "Courier routing and assignment for food delivery service using reinforcement learning". In: *Computers & Industrial Engineering* 164, p. 107871. ISSN: 0360-8352.

Chen, Jing-fang, Ling Wang, Shengyao Wang, Xing Wang, and Hao Ren (2021). "An effective matching algorithm with adaptive tie-breaking strategy for online food delivery problem". In: *Complex & Intelligent Systems*, pp. 1–22.

Chu, Hongrui, Wensi Zhang, Pengfei Bai, and Yahong Chen (2021). "Data-driven optimization for last-mile delivery". In: *Complex & Intelligent Systems*, pp. 1–14.

Cosmi, Matteo, Gaia Nicosia, and Andrea Pacifici (2019a). "Lower bounds for a meal pickup-and-delivery scheduling problem". In: *Proceedings of the 17th Cologne-Twente Workshop on Graphs and Combinatorial Optimization, CTW 2019*, pp. 33–36.

Cosmi, Matteo, Gaia Nicosia, and Andrea Pacifici (2019b). "Scheduling for last-mile meal-delivery processes". In: *IFAC-PapersOnLine*. IFAC-MIM Conference 2019. Vol. 52. 13. Berlin, Germany, pp. 511–516.

Cosmi, Matteo, Gianpaolo Oriolo, Veronica Piccialli, and Paolo Ventura (2019). "Single courier single restaurant meal delivery (without routing)". In: *Operations Research Letters* 47(6), pp. 537–541.

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

Joshi, Manas, Arshdeep Singh, Sayan Ranu, Amitabha Bagchi, Priyank Karia, and Puneet Kala (2021). "Batching and Matching for Food Delivery in Dynamic Road Networks". In: *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE Computer Society: Los Alamitos, CA, USA, pp. 2099–2104.

Joshi, Manas, Arshdeep Singh, Sayan Ranu, Amitabha Bagchi, Priyank Karia, and Puneet Kala (2022). "FoodMatch: Batching and Matching for Food Delivery in Dynamic Road Networks". In: *ACM Trans. Spatial Algorithms Syst.* 8(1). ISSN: 2374-0353.

Klein, Vienna and Claudius Steinhardt (2022). "Dynamic Demand Management and Online Tour Planning for Same-Day Delivery". In: *European Journal of Operational Research*. ISSN: 0377-2217.

Liao, Wenzhu, Liuyang Zhang, and Zhenzhen Wei (2020). "Multi-objective green meal delivery routing problem based on a two-stage solution strategy". In: *Journal of Cleaner Production* 258, p. 120627. ISSN: 0959-6526.

Paul, S., Sunil Rathee, Jose Mathew, and Kranthi Mitra Adusumilli (2020). "An Optimization Framework for On-Demand Meal Delivery System". In: *2020 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pp. 822–826.

Reyes, Damian, Alan Erera, Martin Savelsbergh, Sagar Sahasrabudhe, and Ryan O'Neil (2018). "The Meal Delivery Routing Problem". In: *Optimization-Online*. URL: `http://www.optimization-online.org/DB_HTML/2018/04/6571.html`.

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

Ulmer, Marlin W., Justin C. Goodson, Dirk C. Mattfeld, and Barrett W. Thomas (2020). "On modeling stochastic dynamic vehicle routing problems". In: *EURO Journal on Transportation and Logistics* 9(2), p. 100008.

Ulmer, Marlin W., Barrett W. Thomas, Ann Melissa Campbell, and Nicholas Woyak (2020). "The Restaurant Meal Delivery Problem: Dynamic Pickup and Delivery with Deadlines and Random Ready Times". In: *Transportation Science* 55(1), pp. 75–100.

Xue, Guiqin, Zheng Wang, and Guan Wang (2021). "Optimization of Rider Scheduling for a Food Delivery Service in O2O Business". In: *Journal of Advanced Transportation* 2021.

Zehtabian, Shohre, Christian Larsen, and Sanne Wøhlk (2022). "Estimation of the arrival time of deliveries by occasional drivers in a crowd-shipping setting". In: *European Journal of Operational Research*. ISSN: 0377-2217.