Order Acceptance in Same-Day Delivery

Mathias A. Klapp\textsuperscript{1}, Alan L. Erera\textsuperscript{2}, Alejandro Toriello\textsuperscript{2}

\textsuperscript{1}Engineering School, Pontificia Universidad Católica de Chile
Santiago, Chile
maklapp\textit{at} ing.puc\textit{dot}cl

\textsuperscript{2}H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology
Atlanta, Georgia 30332
\{aerera, atoriello\}\textit{at} isye\textit{dot}gatech\textit{dot}edu

August 3, 2019

Abstract

We study order acceptance dynamics in same-day delivery systems by formulating the Dynamic Dispatch Waves Problem with Immediate Acceptance, which models integrated request management and order distribution for dynamically arriving requests. When a delivery request arrives, a decision is made immediately to accept (offer service) or reject (with a penalty). Accepted requests are not available for immediate dispatch; they must be processed (picked and packed) before they are loaded for delivery in dynamically updated vehicle routes visiting each request’s delivery location by the end of the service day. We consider the case of dispatching a single vehicle from a distribution center, potentially on multiple trips. The objective is to make order acceptance and distribution decisions that minimize the expected sum of total penalties for rejected delivery requests and vehicle travel costs. We develop a framework for dynamic decision policies over continuous time for such systems, where a feasible vehicle dispatch plan is redesigned and used to guide decisions over time. We design methods for determining an initial optimal \textit{a priori} dispatch plan, and for updating it via a heuristic roll-out procedure. Our methods are tested over a family of simulated instances against two common-sense benchmarks and an infeasible relaxed policy that allows the dispatcher to delay a request acceptance decision until its corresponding dispatch time. We estimate the cost-per-request of the best benchmark policy to be 9.7\% higher than our proposed dynamic policy. Conversely, imposing immediate order acceptance on the system results in a cost increase of 4.4\%. 

1 Introduction

Same-day delivery (SDD) is a last-mile order distribution service offered by retailers to customers who make online purchases and expect fast order fulfillment. It has been implemented by e-commerce companies to enhance customer satisfaction, e.g., Amazon, Instacart, Walmart and Google. Nowadays, more than 77 million U.S. residents live in ZIP codes where Amazon offers SDD [23]. It is an example of last-mile delivery services that have helped e-commerce sales grow 16.4% annually and represent 9.5% of U.S. consumer retail sales as of the first quarter of 2018 [29]. According to [34], by offering prompt delivery options e-commerce companies compete with bricks-and-mortar retailers, as they can provide instant customer gratification. However, last-mile delivery is generally the least efficient and most expensive part of the supply chain. Unlike most other parts of the logistics network, it does not scale well, and offers fewer freight consolidation opportunities due to the large variety of SKUs and small volume handled per delivery; according to [22], the last mile can comprise up to 28% of a product’s total transportation cost from its manufacturing location to its final customer. Furthermore, the increasing competitiveness within the e-commerce sector forces service providers to continuously search for last-mile logistics cost reductions.

In contrast to traditional order fulfillment and delivery systems, in SDD the service provider must additionally operate a system where request arrivals, order processing at a fulfillment location, and delivery to the customer location all occur within the operating day. Consequently, SDD operates under a higher degree of information dynamism than other last-mile delivery services with longer response times. In Figure 1a we present a natural next-day delivery setting, where most orders have already been accepted and processed before the delivery operation starts. Conversely, as depicted in Figure 1b, for SDD these operations overlap in time and should be planned simultaneously. Refer to [14, 34] for recent surveys related to city logistics, and to [26, 48] for broader discussions on SDD challenges.

![Figure 1](image-url)  
(a) Next-day delivery  
(b) Same-day delivery  

Figure 1: Description of logistics processes over time for next-day and same-day delivery services

The Dynamic Dispatch Waves Problem (DDWP) [25, 26] seeks to determine a vehicle dispatch plan...
and an order distribution policy in a model where geographically located delivery requests realize according to some stochastic process over time. It assumes that orders are served by a single vehicle on potentially multiple trips from a fulfillment center (depot) and constrains dispatches to be executed within a discrete set of feasible dispatch times (waves). At any wave when the vehicle is available at the depot, it can wait for the next wave to potentially accumulate and load more delivery requests, or can be dispatched to serve a subset of requests ready for service. The routing of the dispatched vehicle incurs travel cost and determines when the vehicle returns to the depot. The problem’s objective is to minimize the expected sum of vehicle travel cost and penalties for unattended requests.

An implicit assumption in the DDWP is that realized orders that will not be served by the vehicle are not formally rejected until the end of the day. Such a setting gives the service provider a degree of flexibility that is likely not available in some practical settings. In this article, we study the Dynamic Dispatch Waves Problem with Immediate Acceptance (DDWP-IA), in which customers are offered an SDD option when placing an order based on delivery location, order timing, and vehicle availability, and if a customer selects the option then an on-time service is guaranteed. To do so, we model a request process as an accept or reject framework: a request is accepted (and must be delivered in the same day), or rejected immediately when received. Such a framework, of course, reduces flexibility and could lead to higher costs. This model can also represent a setting such as Amazon’s same-day delivery service, where a delivery request is not formally rejected and, instead, an automated dispatcher periodically decides whether to offer the SDD option or not. In such a setting our accept/reject framework can preprocess when and where the SDD option is available.

1.1 Contribution

We formulate the DDWP-IA as a semi-Markov decision process (SMDP) [32], which integrates request acceptance with order distribution decision making for SDD systems with a single vehicle. An SMDP generalizes a Markov decision process (MDP) by modeling time as a continuous variable; see [9, 32]. The model studies a complex interrelation, e.g., we save penalties if more orders are accepted earlier in the operating day, but we incur additional travel cost and reduce the flexibility of the dispatch system to accommodate orders that appear later. Conversely, if we leave delivery resources available for the future, we may reject too many orders early in the day and could lead to an under-utilized vehicle. The DDWP-IA also provides an order acceptance framework that benefits from routing economies created by geographic consolidation with previous commitments, e.g., accepting an order with a delivery location close to another
order awaiting its dispatch may only lead to a small marginal increase in travel cost and vehicle travel time. We take advantage of such a framework by designing a policy that proactively searches for these geographic consolidation opportunities, meaning we consider where each planned delivery is located and the additional expected benefits of accepting an order close to a location where it is likely to have a future request realization.

We develop a framework to produce dynamic policies for the DDWP-IA, where a system state is coupled with a state-feasible vehicle dispatch plan (on potentially multiple trips) serving all orders previously accepted and not yet served along with a set of potential future delivery requests that have not yet realized. Specifically, we design a dynamic policy that heuristically rolls-out an optimal \textit{a priori} dispatch plan, which specifies most decisions in advance, and allows only simple plan corrections (recourse rules). Such a plan is used to guide both vehicle dispatch operation and order acceptance decisions. Our modeling approach is related to the “Route-based MDPs” framework [39, 41], but also differs from it by explicitly accounting for potential future delivery requests within the dispatch plan.

We test the performance of our approach in a computational study against two simpler benchmarks: a myopic re-optimization policy that ignores potential future arrivals when making decisions and uses only information about previously accepted orders, and a policy that fixes dispatch waves according to an initial \textit{a priori} solution, but dynamically accepts and assigns orders to dispatches and routes. The cost-per-request of the best benchmark is estimated to be 9.7% higher than our proposed dynamic policy. We also compare our policy to a perfect information bound [10, 35] and to the DDWP in [25], which can delay request acceptance decisions. This last experiment estimates a 4.4% cost increase when imposing immediate order acceptance on SDD systems.

Finally, we remove the assumption of negligible order processing times (from picking and packaging) used in [25] and formulate a model in which a realized request is not immediately ready for delivery; see Figure 2. Our experiments estimate the negative impact of order processing times on the performance of SDD systems.

![Figure 2: Order disclosure time and order ready time for an accepted order.](image-url)
The remainder of this article is organized as follows. Section 2 covers a literature review while Section 3 formulates the DDWP-IA. A lower bounding procedure for the DDWP-IA is described in Section 4. Our solution policies are described in Section 5 and then Section 6 provides a heuristic to speed up the order acceptance mechanism. Finally, Section 7 presents the results of a computational study, and Section 8 provides conclusions.

## 2 Literature Review

The DDWP-IA lies within the broad class of vehicle routing problems (VRP) [15, 38] and is related to the Pickup and Delivery VRP [33] (when all pickups occur at the depot), to the VRP with customer release dates (VRP-RD) [3, 13], and to the Prize-Collecting Traveling Salesman Problem (PCTSP) [7]. In particular, the DDWP-IA is related to the VRP with probabilistic customers (VRP-PC), where a fraction of the orders realize with a given probability after an initial solution is planned. The works in [12, 16, 24, 27, 47] are examples of a priori models of the VRP-PC, which specify solutions in advance and allow for simple recourse rules; refer to [12, 18] for surveys on a priori routing. More complex VRP-PC models are dynamic policies that redesign structural decisions according to newly revealed information over the operating period; see [2, 8, 45, 43, 49] for dynamic VRP-PC models and [28, 30, 36] for surveys on dynamic routing. A subset of dynamic models are rollouts of a priori policies that have proved successful in stochastic routing problems [19, 20, 21, 42].

The DDWP-IA differs from VRP-PCs, which are mainly focused on the order distribution component and were primarily conceived for order pickup and/or generic goods operations. In the case of generic goods, such as transportation of commodities, vehicles can be preloaded to account for future order arrivals and the dispatcher can dynamically insert new customer visits enroute. In contrast, the DDWP-IA includes request acceptance, order processing times, and assumes customer-specific deliveries, which are more common in SDD.

The distribution system within the DDWP-IA is closely related to dynamic last-mile delivery routing problems. Some examples of such problems are the DDWP [25], The Delivery Dispatching Problem [46], The Dynamic Multiperiod Routing Problem (DMPRP) [2, 44], The Same Day Delivery Problem for Online Purchases [45, 48] and other state of the art research efforts; refer to [25] for a detailed literature review. These problems model a depot with dynamic realization of delivery requests served via delivery vehicle routes. Compared to VRP’s, these problems typically have vehicles execute multiple trips per day [4, 6].
Also, some explicitly model order release dates at the depot [3, 13].

The Home Delivery Problem (HDP) [11] combines request management and routing decisions. It models a dynamic time slotting process for next-day grocery delivery requests, where customers dynamically place requests one day before delivery. Simultaneously, a dispatcher determines in real-time whether to accept each request or not and, if so, assigns a time slot for next-day delivery. The following day, a set of routes is designed covering all accepted requests and taking into account promised delivery time windows. This research effort studies the interaction between promise of service and available next-day dispatch capacity. A related tactical time slot management problem for distribution operations is addressed in [1]. Both studies differs from SDD problems, where request acceptance and order delivery decisions must be executed simultaneously in the same day.

In [43] the authors present a dynamic VRP-PC modeling a single vehicle operation, where new customer pickup requests can dynamically be included while the vehicle is enroute. Each time the vehicle arrives at a particular location, the dispatcher chooses a subset of all newly realized requests to accept within the current route plan. The model seeks to maximize the expected number of customers served. The paper’s solution policy is designed via Approximate Dynamic Programming; specifically, the authors use value function approximation (see [31]) and state space aggregation techniques. The value-to-go function in a given state after an acceptance decision is estimated via offline simulation and assumed to depend on aggregated state information, namely the amount of free time left in the route before the end of the operation and the current decision time. The DDWP-IA differs from a pickup problem and works with customer specific orders to be picked up at a depot and delivered to the customer’s location. This difference implies specific order release times at the depot determining the earliest possible time to load the order into a vehicle. Also, the DDWP-IA works in continuous time and makes immediate order acceptance decisions each time a request arrives.

A VRP-PC with online request realizations and immediate request acceptance decisions is presented in [5]. All accepted orders are served with multiple dispatches per day of a single vehicle. A solution to this model does relatively short dispatches imposed by service time windows and route duration constraints that induce vehicle returns to the depot. Therefore, it applies to settings in which a delivery must occur within a short period of time from the order’s request; i.e., delivery of perishable goods such as meals or one-hour services. In contrast, our model treats each route duration as an unconstrained decision to be optimized subject to order realization times at the depot. To execute request acceptance decisions, a scenario-based planning approach similar to [8] is used to heuristically estimate an order’s insertion profit over multiple simulated
future order scenarios. Our request acceptance mechanism does not require simulation and proactively plans returns to the depot, balancing future reaction capabilities and routing costs.

Finally, a same-day delivery routing problem with dynamic service pricing decisions is proposed in [40]. When a delivery request arises in this setting, a decision maker dynamically defines a price for each available delivery option, e.g., same-day, next-day, two-day. The customer then observes offered services and prices, and chooses the option that maximizes his surplus depending on a willingness to pay for each delivery service type. Once a service is chosen, it must be delivered to the customer’s destination on time. Accordingly, dynamic routing decisions are taken heuristically based on the author’s previous work; see [44, 43]. Compared to this pricing-based mechanism, our model assumes a given stochastic demand for service, in which the decision maker directly chooses which orders to accept; it is simpler, but does not require a detailed customer behavioral model or estimate its willingness to pay for each delivery service option.

3 Problem formulation

The DDWP-IA models a depot (node 0) and its service area defined by a finite set of geographic customer locations $I := \{1, \ldots, |I|\}$, representing neighborhoods, city blocks or delivery lockers; let $E$ be the set of edges (road network) between all pairs in $I \cup \{0\}$. Traversing an edge $e \in E$ takes $d_e$ time and costs $\gamma d_e$; assume for simplicity that time and cost values are proportional to each other, non-negative, and that they satisfy the triangle inequality.

The operating day is modeled as a continuous set $\mathcal{T} = [T, 0]$; where time is counted backwards as a resource being depleted so that $t = T$ represents the start of the operating day. The day is also discretized into $W$ possible vehicle dispatch times (waves). Each wave is a decision epoch when a vehicle (if available at the depot) can be loaded and dispatched from the depot to serve a subset of orders, or wait for the next wave. In practice, these wave times are chosen based on many factors, including constraints associated with driver shifts and efficiencies gained by organizing the warehouse in order picking waves [17]. Let $\mathcal{W} := \{W, \ldots, 1\}$ be the discrete set of waves, i.e., feasible vehicle dispatch points, such that each wave $w \in \mathcal{W}$ occurs at time $t_w \in \mathcal{T}$ with $T = t_W > t_{W-1} > \ldots > t_1 > t_0 = 0$; 0 represents the terminal wave at day’s end. Define the upcoming waves at $t$ as $\mathcal{W}(t) := \{w \in \mathcal{W} : t_w \leq t\}$; similarly let $\mathcal{W}_0(t) := \mathcal{W}(t) \cup \{0\}$.

Customer delivery requests arrive over time according to a stochastic counting process $N_i(t) \in \mathbb{Z}_+, t \in \mathcal{T}$, defined for each $i \in I$. We assume arrivals are independent between locations, stopping after a service
cut-off time $t^{ct}$, and satisfying the Markovian property. An example is a Poisson arrival process with order arrival rate $\lambda_i$ per time unit, truncated after $t^{ct}$.

An acceptance decision must occur immediately after a delivery request from $i \in I$ arrives at time $t \in [T, t^{ct}]$. If accepted, the new order must be processed at the depot in $p \geq 0$ time units and then delivered to the customer after its release time from the depot at $t - p$ in a vehicle dispatch at a wave $w \in W(t - p)$. Any rejected delivery request to location $i$ is lost at a penalty cost $\beta_h > 0$.

Dispatch decisions may occur when the vehicle is available at the depot at times $t_w$, for some wave $w \in W$. A vehicle dispatch executing a delivery route $r = \{0, i_1', \ldots, i_r', 0\}$ with $m_r$ location visits has a transportation cost $\gamma d(r) := \gamma \sum_{j=1}^{m_r+1} \{d(i_j', i_{j+1})\}$ and spends $t(r) := d(r) + \sum_{j=0}^{m_r} u_{i_j}$ time units, where $u_{i}, i \in I$ is the service time at location $i$, assumed to be independent of the number of requests served at $i$; this models an urban delivery situation in which parking and access times dominate service times. The value $u_0$ represents a vehicle set-up time at the depot. The vehicle returns to the depot at time $t_w - t(r)$ and becomes available for dispatch again at wave $q(w, r) := \max\{k \in W_0(t_w - t(r))\}$. We only consider elementary routes, since we do not increase travel time if we consolidate all requests at node $i$ into one visit. Also, we do not consider split deliveries without loss of optimality, i.e., a route serves all accepted and released requests at any visited location.

The system’s state is described at any time $t \in T$ by $s = (t, a, w) \in S$, where $S$ is the set of all feasible states. The parameter $a$ is the vector of open commitments, and each component indexed by $i \in I$ indicates the earliest wave $a_i \in W$ in which a vehicle dispatch visiting $i$ can cover all its pending accepted orders. No orders are waiting for service at $i$ when $a_i = \infty$. Given $a$, let $I_a := \{i \in I : a_i < \infty\}$ be the subset of nodes with pending orders. The parameter $w \in W_0(t)$ represents the earliest upcoming wave when the vehicle is available for dispatch at the depot. A state $s$ does not carry disaggregated information regarding specific orders at $i$ because all such orders can be covered in one visit dispatched no earlier than $\min(w, a_i)$ without loss of optimality.

A state $s$ is feasible if there exists a dispatch plan that feasibly serves all commitments in $a$. A dispatch plan $\pi = \{r^\pi_k : k \in W^{\pi}\}$ is defined by a set of elementary routes $r^\pi_k$ indexed by its set of dispatch waves $W^{\pi} \subset W$. Formally, for $t \in T, w \in W_0(t)$, a state $s = (t, a, w)$ is feasible if and only if there exists a dispatch plan $\pi$ satisfying:

1. Plan $\pi$ starts operating after wave $w$, i.e., $k \leq w$ for each $k \in W^{\pi}$.

2. Plan $\pi$ covers all commitments, i.e., if $i \in I_a$, then there exists $k \in W^{\pi}$ such that $k \leq a_i$ and $i \in r^\pi_k$. 

8
3. Routes in $\pi$ do not overlap in time, i.e., for any two consecutive dispatches $k^+ > k^-$ in $\mathcal{W}/\pi$ we have that the next available wave with the vehicle available at the depot after $k^+$ occurs no later that $k^-$:

$$q(k^+, r^w_{k^+}) \geq k^-.$$ 

A plan’s feasibility condition only depends on $a$ and $w$ and is preserved over time until the earliest possible dispatch wave $w$; if $\pi$ is feasible for state $(t, a, w)$, then it is also feasible for any state $(t', a, w) : t' \in [t_w, t]$. This is a useful property in heuristic design. To check a plan’s feasibility, we must solve a VRP with release dates [3, 13]. A special case occurs for $p = 0$, when all pending orders are released for dispatch on or before wave $w$ and any feasible state $s$ has a feasible plan $\pi$ consisting of a single vehicle route dispatched after $w$ covering all commitments. Let $\mathcal{S}$ be the set of all feasible states.

### 3.1 Actions, transitions and costs

An accept/reject decision is immediately taken after a request realizes at location $i$ in state $s = (t, a, w)$. Rejecting an order costs $\beta_i$, but keeps the system’s state unaltered. Accepting it is free of charge, but can only be performed if the post-decision state $(t, a^i, w)$ remains feasible, where $a^i$ is the updated vector of commitments. Denote the order’s earliest dispatch wave from the depot as $b := \max\{k \in \mathcal{W}(t-p)\}$; then $a^i$ is defined as $a^i_j = a_j$ for $j \neq i$ and $a^i_i = \min\{a_i, b\}$. 

A dispatch decision is taken when time matches the next dispatch wave at any state $(t_w, a, w)$. If we restrict ourselves to optimal Traveling Salesman Problem (TSP) routes, a vehicle dispatch is fully determined by a subset $Q \subseteq I$ of node visits representing an optimal tour over $Q \cup \{0\}$, minimizing travel time and simultaneously maximizing the return wave $q^*(w, Q)$. Any feasible dispatch $Q$ at wave $w$ keeps the system in a feasible post-decision state $(t, a(w, Q), q^*(w, Q))$, where $a(w, Q)$ is the updated vector of commitments defined as $a(w, Q)_i = a_i$ when $i \notin Q$ and $\infty$ otherwise. $Q = \emptyset$ is the special action that represents waiting at the depot at zero cost and sets $q^*(w, \emptyset) = w - 1$. Figure 3 depicts a flowchart of possible system transitions and actions related to state $(t, a, w)$.

![Flowchart of system transitions and actions in state $(t, a, w)$ for the DDWP-IA](image)
3.2 Dynamic programming model

We model the DDWP-IA as an SMDP. Given a time \( t \in T \), let

\[
\phi(i,t',t) := P\{( \tau_i = t' ) \cap ( \tau_j > \tau_j, \forall j \neq i | \tau_i < t )
\]

be the probability density of the next order arriving at time \( t' \in [t, t'] \) at location \( i \in I \), and let

\[
\psi(t',t) := \prod_{i \in I} P(\tau_i < t' | \tau_i < t)
\]

be the probability that no order arrives between \( t \) and \( t' \). In the particular case of the Poisson process we have \( \phi(i,t',t) = \lambda_i e^{t' - t} \) and \( \psi(t,t') = e^{(\sum_{j \in I} \lambda_j)(t - \max(t, t'))} \). Let \( C(s) \) be a function representing the optimal expected cost-to-go at state \( s \), and let \( C^*(a^0) := C(T, a^0, W) \) be the optimal expected cost with a vector \( a^0 \) of commitments accepted before the operation starts, typically all ready at wave \( W \). The SMDP (1) computes \( C^*(a^0) \) recursively over time

\[
C(0, \infty, 0) = 0, \quad (1a)
\]

\[
C(t_w, a, w) = \min_{Q \subseteq T; (t, a(w), Q) \in \mathcal{S}} \left\{ \gamma d^*(Q) + C(t_w, a(w), Q, q^*(w, Q)) \right\}, \quad \forall (t_w, a, w) \in \mathcal{S} \quad (1b)
\]

\[
C(t, a, w) = \psi(t, t_w)C(t_w, a, w) + \sum_{i \in I} \int_{t' = t_w}^{t'} \phi(i, t', t) \tilde{C}(t', a, w, i) dt', \quad \forall (t, a, w) \in \mathcal{S} : t > t_w, \quad (1c)
\]

\[
\tilde{C}(t, a, w, i) = \min\{ \beta_i + C(t, a, w), C(t, a', w) : (t, a', w) \in \mathcal{S} \}, \quad \forall i \in I, (t, a, w) \in \mathcal{S} : t > t_w, \quad (1d)
\]

where Equation (1a) sets the terminal cost equal to zero. Equation (1b) models the state transition during a dispatch decision and states that the cost-to-go at state \( (t_w, a, w) \) is equal to the minimum sum of dispatch cost \( \gamma d^*(Q) \) plus the post-decision cost-to-go \( C(t_w, a(w), Q, q^*(w, Q)) \) over all feasible dispatch subsets \( Q \subseteq I \).

Equation (1c) models the evolution of the system over time and states that any cost-to-go \( C(t, a, w) : t > t_w \) is equal to the cost-to-go in the next dispatch decision \( C(t_w, a, w) \) if no orders arrive between \( t \) and \( t_w \) (with probability \( \psi(t, t_w) \)), or equal to \( \tilde{C}(t', a, w, i) \) if the next order realizes at node \( i \) and time \( t' \in [t_w, t] \) (with probability density \( \phi(i, t', t) \)); \( \tilde{C}(t, a, w, i) \) represents the cost-to-go immediately before the acceptance decision defined in Equation (1d), equal to the minimum cost-to-go between rejecting the order \( C(t, a, w) + \beta_i \) and accepting it \( C(t, a', w) \) (if feasible).

Model (1) is intractable; it has an uncountable state space, exponentially many dispatch decisions for each state, and an uncountable number of terms in the expectations that model transitions in time. Also, it
is NP-Hard to evaluate \( d^*(Q) \) and \( q^*(w,Q) \), which involve solving a TSP over \( Q \cup \{0\} \). In the next Section, we develop approximate solutions to the DDWP-IA.

4 The deterministic DDWP and lower bounds

We first derive a perfect information lower bound by solving the simplified problem where the number of delivery requests and their arrival times at customer locations \( i \in I \) are disclosed before the operation starts. In this setting each arrival counting function \( N_i(t) \in \mathbb{Z} \) at location \( i \in I \) up to time \( t \in \mathcal{T} \) is fully known and all relevant information is available to plan request acceptance and vehicle dispatch decisions before the operation starts. In the deterministic case, the DDWP-IA model collapses to a deterministic variant of the DDWP solved in [25] via branch and cut approaches for routing problems.

In the deterministic DDWP it is still infeasible to serve an order before its release time, meaning that the maximum number of orders that can be accepted and dispatched to location \( i \) by wave \( w \) is at most \( n_{i,w} := N_i(t_w + p) \). Without loss of optimality, a plan visiting a node \( i \) in a vehicle dispatch at wave \( w \) covers all \( n_{i,w} \) requests and, therefore, it completely defines all accepted orders. Figure 4 provides an example where functions \( N_i(t) \) and \( n_{i,w} \) are depicted for a particular location \( i \).

![Figure 4: Evolution of requests disclosed (N_i(t)) and orders released (n_{i,w}) over time for a particular realization (\omega) of arrivals at node i in a seven-wave horizon, a cutoff time t^{ct} = t_2, and a processing time p.](image)

Define \( h(i,j) = \min\{w \in \mathcal{W} : t_w \geq t_{(i,j)} + u_i + u_j + \mathbb{I}_{(i+j>0)}(u_0) + \mathbb{I}_{(i>0)}(t_{(0,i)}) + \mathbb{I}_{(j>0)}(t_{(0,j)})\} \) as the latest possible dispatch wave for \( \{i,j\} \in E \), and \( I_w := \{i \in I : w \geq h(0,i)\} \subset I \) and \( E_w := \{e \in E : w \geq h(e)\} \subset E \) for each \( w \in \mathcal{W} \) as the sets of feasible nodes and edges for a vehicle dispatch at wave \( w \). Also, define the cut set \( E_w(S) = \{\{i,j\} \in E_w : i \in S, j \not\in S\} \), for any subset \( S \subseteq I_w \), and define \( t_e = t_e + 0.5u_i + 0.5u_j \) as the adjusted time spent at edge \( e = \{i,j\} \in E \) considering service times. The Integer Program in (2) solves the
deterministic DDWP,

$$C^D(a^0, N(t)) = \min_{\{x, y, x, s\}} \sum_{i \in I} B_i (n_i, 0) z_i + \sum_{w=1}^W (n_i, 0 - n_i, w) y^w_i + \sum_{w=1}^W \sum_{e \in E_w} \gamma d_e x^w_e$$  \hspace{1cm} (2a)

s.t.  \hspace{1cm} \sum_{w=h(i,0)} d_i^0 y^w_i = 1, \hspace{1cm} \forall i \in I^0  \hspace{1cm} (2b)

$$z_i + \sum_{w=h(i,0)} y^w_i = 1, \hspace{1cm} \forall i \in I  \hspace{1cm} (2c)$$

$$\sum_{e \in E_w(0)} x^w_e \leq 2, \hspace{1cm} \forall w \in \mathcal{W}  \hspace{1cm} (2d)$$

$$\sum_{e \in E_w(0)} x^w_e \geq 2y^w_i, \hspace{1cm} \forall w \in \mathcal{W}, \forall S \subseteq I_w, \forall i \in S  \hspace{1cm} (2e)$$

$$\sum_{e \in E_w(S)} x^w_e \leq \sum_{k<w} (t_w - t_k) v^w_k, \hspace{1cm} \forall w \in \mathcal{W}  \hspace{1cm} (2f)$$

$$\sum_{k<w} s_k + \sum_{k<w} v^w_k = 1, \hspace{1cm} \forall w \in \mathcal{W} \setminus \{W\}  \hspace{1cm} (2g)$$

$$\sum_{k<w} v^w_k = \sum_{k>w} v^w_k + s_w, \hspace{1cm} \forall w \in \mathcal{W} \setminus \{W\}  \hspace{1cm} (2h)$$

$$v^w_k \in \{0, 1\}, \hspace{1cm} \forall w, k \in \mathcal{W}_0 : k < w  \hspace{1cm} (2i)$$

$$s_k \in \{0, 1\}, \hspace{1cm} \forall k \in \mathcal{W}_0 : k < W  \hspace{1cm} (2j)$$

$$z_i \in \{0, 1\}, \hspace{1cm} \forall i \in I  \hspace{1cm} (2k)$$

$$y^w_i \in \{0, 1\}, \forall i \in I_w, \text{ and } x^w_e \in \{0, 1, 2\}, \forall e \in E_w, \hspace{1cm} \forall w \in \mathcal{W}  \hspace{1cm} (2l)$$

where variable $z_i$ is equal to 1 if node $i$ isn’t visited, and 0 otherwise; $y^w_i$ is equal to 1 if a dispatch at wave $w$ visits node $i$, and 0 otherwise; $x^w_e$ is equal to $m \in \{0, 1, 2\}$ if the vehicle traverses edge $e$ $m$ times at a dispatch in wave $w$; $v^w_k$ is equal to 1 if a dispatch at $w$ returns at wave $k$, and 0 otherwise; and $s_k$ is equal to 1 if the vehicle waits at the depot until wave $k$, and 0 otherwise ($s_0 = 1$ implies an empty plan with no dispatch throughout the horizon). The objective function (2a) minimizes the sum of total vehicle travel costs plus penalties for rejected requests. Constraints (2b) force all initially accepted visits in $a^0$ and (2c) guarantee visiting each node $i$ at most once at wave $w$. Constraints (2d) - (2e) guarantee that vector $x^w$ defines a feasible tour only visiting nodes selected by the vector $y^w$. Constraints (2f) force routes to satisfy duration limits determined by $v^w_k$. Finally, wave flow constraints (2g)-(2h) enforce vehicle conservation throughout time. We implicitly use two properties proved in [25] that any feasible solution satisfies without loss of optimality: (1) Any location $i \in I$ is visited by the vehicle at most once. (2) The vehicle does not wait after the first dispatch.
Problem (2) generalizes the Prize-Collecting TSP (PC-TSP) and its size only depends on time through the number of waves \( W \). We will use it to compute a perfect information relaxation (PIR) that computes a different optimal solution for each scenario realization of the random parameters. To simplify upcoming notation, we say that any vector of variables \((x, y, z, v, s)\) representing a feasible vehicle dispatch plan starting after wave \( W \), i.e., satisfying constraints (2c) through (2l), belongs to domain \( \mathcal{D}(W) \).

5 Solution policies for the stochastic case

We next develop a solution framework for the DDWP-IA based on approximate dynamic programming (ADP) [31, 39]. Our model maintains a system state \( s \) coupled with a feasible dispatch plan \( \pi \) serving all accepted and pending delivery requests in \( s \) along with a set of potential future delivery requests that have not yet realized. This dispatch plan is used to guide upcoming order acceptance and vehicle dispatch decisions, and it is dynamically updated when new information becomes available. Each policy \( P \) constructs an initial dispatch plan \( \pi \) that is feasible for the initial state \( s^0 \), defined by routes \( r^\pi_k \) dispatched at waves \( k \in \mathcal{W}^\pi \), each visiting a subset \( Q^\pi_k \subseteq I \) of nodes. After the operation starts, policy \( P \) dynamically updates \( \pi \) to keep it feasible throughout all states \( s^j \in \mathcal{S}, j = \{0, 1, 2, 3, 4, \ldots\} \) visited by the system. In the dynamic routing literature, models like ours that carry a route plan with the system’s state space are sometimes referred to as route-based Markov decision processes, [39, 41].

Define the random list of online request arrivals as \( \mathcal{L} \), where each request is completely defined by its arrival time and delivery location. Algorithm 1 provides a high-level pseudo-code description to compute the cost \( C^P(a^0, \mathcal{L}) \) of a policy \( P \) given a list of realized requests \( \mathcal{L} \) and \( a^0 \). \( P \) is determined by three functions used to update the plan: \textbf{IniPlan}, \textbf{ArrivalUpdate} and \textbf{DispatchUpdate}.

Algorithm 1 initializes the state of the system in line 2 and calls function \textbf{IniPlan}, which constructs an initial plan \( \pi \) in line 3. Then, it runs an event-based simulation that advances to the time of the next event in line 5. If it is a request arrival event, it updates plan \( \pi \), calling \textbf{ArrivalUpdate} in line 8; if the updated plan covers the new request it accepts it in line 9 and updates the state \( s \); otherwise, it rejects it and pays the penalty cost in line 10. On the other hand, if the event is a dispatch decision, it updates the plan, calling \textbf{DispatchUpdate} in line 12 and, if \( w \) belongs to the set of planned dispatches, it dispatches route \( r^\pi_w \) and executes all corresponding cost, state, and plan updates in line 14; otherwise, the vehicle waits at the depot one wave (line 15). The computing time of \textbf{ArrivalUpdate} is critical to allow fast acceptance
Algorithm 1 Implementation of a generic policy $P$

1: procedure EXECUTEPOLICY($P, a^0, L$)
2: Initialize cost and state: $C \leftarrow 0, s := (t, a, w) \leftarrow (T, a^0, W)$
3: Initialize plan: $\pi \leftarrow \text{INIPLAN}(P, a^0)$
4: while (unprocessed requests ($L \neq \emptyset$) or waves left ($w > 0$)) do
5:  Update time $t$ to next event
6:  if (next event is a request arrival) then
7:    Pull out request from $L$, get its location $i$ and release wave $b \leftarrow \max\{x \in \mathcal{W}_0(t - p)\}$
8:    Update plan: $\pi \leftarrow \text{ARRIVALUPDATE}(P, \pi, s, i, b)$
9:    if (plan $\pi$ covers $i$ after wave $b$) then accept request: $a_i \leftarrow \min\{a_i, b\}$
10:   else reject: $C \leftarrow C + \beta_i$
11:  else (next event is a dispatch decision) then
12:    Update plan: $\pi \leftarrow \text{DISPATCHUPDATE}(P, \pi, s)$
13:    if plan $\pi$ dispatches at wave $w$ then
14:      dispatch route $r^\pi_w$: $C \leftarrow C + \gamma(r^*_w), a \leftarrow a(w, Q^\pi_w), w \leftarrow q(w, r^\pi_w)$, remove $r^\pi_w$ from $\pi$
15:    else wait at the depot: $w \leftarrow w - 1$
16: return $C$

decisions. We next define multiple policies that differ in how they implement IniPlan, ArrivalUpdate and DispatchUpdate.

5.1 Myopic policy

The first policy ignores all available probabilistic information regarding future request arrivals, but makes optimal decisions with respect to the information disclosed so far. When a new request realizes at state $(t, a, w)$ at location $i$ with release wave $b$, the myopic policy (MP) solves the IP defined in (3) and outputs its optimal dispatch plan in function $\text{ArrivalUpdate}(MP, \pi, s, i, b)$

$$\min_{\{x, y, z, v, s\} \in \mathcal{D}(w)} \beta_i\{z_i + \sum_{k=\max(b+1, h(0, i))}^w y^k_i\} + \sum_{k=1}^w \sum_{e \in E_k} \gamma_d x^k_e \tag{3a}$$

s.t. $\sum_{k=h(0,i)}^w y^k_i = 1, \quad \forall i \in I_a. \tag{3b}$

The objective (3a) minimizes vehicle travel cost plus a penalty paid if the solution does not cover the new request. The plan is forced to be feasible, i.e., $\{x, y, z, v, s\} \in \mathcal{D}(w)$ and constraints (3b) guarantee the coverage of all previous commitments. For the Myopic policy, $\text{DispatchUpdate}(MP, \pi, s)$ does not alter the previous plan $\pi$. Finally, function $\text{IniPlan}(MP, a^0)$ determines an initial plan $\pi$ with a single route equal to an optimal TSP route over $I_a \cup \{0\}$ dispatched at the latest possible wave.
A myopic plan tends to build one single and long dispatch route, leaving few recourse possibilities (or none). It focuses on consolidating all accepted orders, but does not consider rejecting potential future requests. If we are interested in a myopic solution with multiple returns to the depot (more recourse), we can heuristically set a maximum route duration $d_{\text{max}}$ to enforce this behaviour in (3). The value of parameter $d_{\text{max}}$ must be calibrated beforehand.

### 5.2 A priori policy

Now we present an *a priori* policy (AP) in which a static dispatch plan $\pi$ is determined before execution, using all probabilistic information available at time $T$. During operation, it accepts each request released at a wave and location covered by the plan. As in [25], we plan an optimal *a priori* dispatch plan in which no recourse actions are allowed. The travel cost and expected penalty cost of such a plan is known at time $T$. Define $\bar{n}_i^w := \mathbb{E}(N_i(t_w + p))$ as the expected number of requests realized at node $i$ and released by wave $w$; for a Poisson process with rate $\lambda_i$ we have $\bar{n}_i^w = \lambda_i \max(0, T - \max(t^{ct}, t_w + p))$; an example is depicted in Figure 5.

**Figure 5:** Expected orders realized at time $t$ ($\mathbb{E}(N_i(t))$) and expected orders released for delivery to node $i$ at wave $w$ ($\bar{n}_i^w$) for a particular Poisson arrival process with $t^{ct} = t_2$ and $\lambda_i = 0.8$.

If the latest planned visit to location $i$ is at wave $w$, then the total expected penalty cost paid for location $i$ at time $T$ is $\beta_i(\bar{n}_i^0 - \bar{n}_i^w)$, independent of earlier visits to $i$. So, AP is equivalent to solving a deterministic DDWP instance with $\bar{n}_i^w$ orders released at any node $i$ and wave $w$. The IP that solves for this optimal *a priori* plan is

\[
\begin{align*}
\min_{\{x,y,z,v\} \in \mathcal{P}(W)} & & \sum_{i \in I} \sum_{w=1}^{W} \beta_i (\bar{n}_i^0 z_i) + \sum_{w=1}^{W} (\bar{n}_i^0 - \bar{n}_i^w) y_i^w \text{ s.t. } & & \sum_{w=1}^{W} y_i^w = 1, \forall i \in I_0; \quad (4a) \\
& & \sum_{w=1}^{W} v_i^w = 0, \forall i \in I_0; \quad (4b)
\end{align*}
\]
it shares its feasible region with (2), but has an objective determined by expected future request rejections. The function $\text{IniPlan}(AP, a^0)$ returns a dispatch plan solving (4); function $\text{ArrivalUpdate}(AP, \pi, s, i, b)$ produces no change to the plan; and $\text{DispatchUpdate}(AP, \pi, s)$ improves the performance of the plan at each wave $w \in \mathcal{W}^\pi$ in state $s$ by skipping from the upcoming dispatch all planned visits without pending orders released by $t_w$.

### 5.3 Myopic policy with fixed a priori dispatch

We next present a myopic policy (MPF) that predetermines at time $t = T$ a subset of dispatch waves based on the optimal a priori dispatch plan. Intuitively, MPF may outperform AP through re-optimization and correct MP’s myopic dispatch structure. Let $\mathcal{W}^{AP}$ be the waves used by a solution to (4). At any state $s$ visited by the system, MPF keeps a feasible dispatch plan $\pi$ to $s$ that only plans dispatches at waves in $\mathcal{W}^{AP}$. The plan $\pi$ is initialized calling function $\text{IniPlan}(MPF, a^0)$, with an optimal a priori plan. When a request arrives at state $(t, a, w)$ and node $i$ with release wave $b$ the plan calls function $\text{ArrivalUpdate}(MPF, \pi, s, i, b)$, which solves

$$
\min_{\{x, y, z, v, s\} \in \mathcal{D}(w)} \frac{\beta}{\{z_i + \sum_{k=\max(b+1, h(0,i))}^w y_i^k\} + \sum_{k=1}^w \sum_{e \in E_k} \gamma d_e \chi^k} \\
\text{s.t.} \quad \sum_{k=h(0,i)}^w y_i^k = 1, \quad \forall i \in I_a, \quad (5b)
$$

$$
v_w^k = 0, \quad \forall w, k \in \mathcal{W} : k < w, w \notin \mathcal{W}^{AP}. \quad (5c)
$$

The problem is similar to (3), but adds constraints (5c) to ban waves not used in the a priori solution; function $\text{DispatchUpdate}$ leaves the plan unaltered. A constrained dispatch policy such as MPF emulates and improves a system that practitioners may use: a fixed dispatch policy with a myopic plan update. It builds an initial dispatch structure based on probabilistic information and, once the daily operation starts, the decision maker assign requests to available dispatch time slots myopically.

### 5.4 Full rollout of the a priori policy

A better but more involved idea is to fully roll out the optimal a priori policy (RP) and re-optimize (4) at each state $s$ visited by the system, after new information arrives when a request realizes and as expected arrivals considered in the plan do not appear. Thus, we will re-optimize the plan when an order arrives, and before each planned dispatch decision. The initial dispatch plan $\pi$ for RP matches the optimal a priori plan. Define the expected number of requests realized after time $t \in \mathcal{T}$ at node $i \in I$ that are ready by wave
for a Poisson process with rate $\lambda_i$ this becomes $f_{i,w}(t) = \lambda_i \max\{0, t - \max(t^{ci}, t_w + p)\}$. RP will update the dispatch plan in function $\text{ArrivalUpdate}(RP, \pi, s, i, b)$ solving an optimal a priori plan for state $s$ in (7), conditioned on a new request realization at node $i$,

\[
\min_{\{x,y,z,v\} \in \mathcal{D}(w)} \sum_{k=1}^{w} \sum_{\mathcal{E}} \gamma dx_k^j + \sum_{j \in I} \beta_j \left\{ f_{j,0}(t)z_j + \sum_{k=h(0,j)}^{w} \left( f_{j,0}(t) - f_{j,k}(t) \right) y_k^j \right\}
\]

s.t. \[\min \{a_j,w\} \sum_{k=h(j,0)}^{w} y_k^j = 1, \quad \forall j \in I_a. \] (7a)

where the problem’s domain equals (3), but it incorporates penalties for expected rejections of future order arrivals plus an extra penalty for rejecting the realized request at location $i$ in the objective (7a). We save some computational effort by skipping the re-optimization of the plan if the new request is already covered by the previous plan. Finally, we update the plan in function $\text{DispatchUpdate}(RP, \pi, s)$ solving

\[
\min_{\{x,y,z,v\} \in \mathcal{D}(w)} \sum_{k=1}^{w} \sum_{\mathcal{E}} \gamma dx_k^j + \sum_{j \in I} \beta_j \left\{ f_{j,0}(t)z_j + \sum_{k=h(0,j)}^{w} \left( f_{j,0}(t) - f_{j,k}(t) \right) y_k^j \right\}
\]

s.t. \[\min \{a_j,w\} \sum_{k=h(j,0)}^{w} y_k^j = 1, \quad \forall j \in I_a. \] (8a)

Computing RP may require the solution of an IP when a request arrives and before each dispatch wave in the horizon, and these IP’s will grow in difficulty as $I$, $W$ and arrival frequency per node grow. To speed up computation, we warm-start the incumbent solution of an IP with the latest feasible plan available from previous plan re-optimizations. Also, we keep all subtour elimination cuts from previous IPs sharing the same network structure. Finally, we do not solve each problem to optimality and set an optimality tolerance (0.5%) and maximum solution time (1800 seconds). We evaluate all policies, including this one, via computational experiments in Section 7. Particularly, the computational effort in $\text{ArrivalUpdate}$ is critical and motivates us to design a heuristic rollout policy in Section 6.
6 A generic heuristic

Now, we propose a heuristic to improve any plan \( \pi \) feasible for the generic IP

\[
GC(w, g) = \min_{(x, y, z, s) \in \mathcal{P}(w)} \sum_{k=1}^{w} \sum_{e \in E_k} \gamma d_{e,k} + \sum_{i \in I} \left\{ g_{i,0}z_i + \sum_{k=h(0,i)}^{w} g_{i,k}y_i \right\},
\]

where \( w \) is the earliest dispatch wave and \( g_{i,k} > 0 \) represents any cost for serving node \( i \) at wave \( k \); case \( k = 0 \) represents no service. All IPs defined in Sections 4 and 5 can be stated in this form.

We run multiple neighborhood searches (NS) over a dispatch plan, each exploiting the wave structure of a plan and solving multiple instances of PC-TSPs that arise from partial plan optimizations. We extend the local search procedure from [25], by adding two improvements: we randomly destroy solutions to avoid locally optimal plans and use randomized acceptance rules to evaluate a candidate solution; see Appendix A.2. Second, we solve each PC-TSP with a heuristic defined in Appendix A.3.

Algorithm 2 Heuristic Search Procedure

1: \textbf{procedure} RUNHEURISTIC(Initial feasible plan \( \pi^0 \), maximum random destructions \( k^{\text{max}} \))
2: \hspace{1em} \( \pi \leftarrow \text{Copy}(\pi^0) \), \( \pi^* \leftarrow \text{Copy}(\pi^0) \)
3: \hspace{1em} \textbf{do}
4: \hspace{2em} \textbf{if} \ (-\text{INTRA}_L \text{S}(r, r^*) \text{ and } -\text{INTER}_L \text{S}(r, r^*) \text{ and } -\text{WAVES}_L \text{S}(r, r^*)) \text{ then}
5: \hspace{3em} \text{RANDOMDESTRUCTION}(r)
6: \hspace{1em} \textbf{while} (less than \( k^{\text{max}} \) passes)
7: \hspace{1em} \textbf{return} \( r^* \)

Algorithm 2 provides a high-level description of the heuristic; it requires an initial feasible plan \( \pi^0 \), and uses three neighborhood searches to improve upon the best available plan \( \pi^* \): (1) intra-route local search (\text{Intra}_L \text{S}), \textit{i.e.}, single route node selection and re-sequencing; (2) inter-route local search (\text{Inter}_L \text{S}), \textit{i.e.}, node exchanges between routes and re-sequencing; and (3) wave local search (\text{Waves}_L \text{S}), \textit{i.e.}, changes in the number of routes and dispatch times. Each local search returns \textit{true} if it has updated and accepted a new local solution \( \pi \), and else, it returns \textit{false}. If no local search update is made, we run a solution destruction operator that randomly deletes a percentage of the nodes and routes in plan \( \pi \); this is done \( k^{\text{max}} \) times before the heuristic outputs plan \( \pi^* \). The details of all functions can be found in Appendices A.1 and A.2.

We make a final improvement to the meta-heuristic by running two local search moves in series before evaluating a candidate plan’s cost. Specifically, we run function \textit{RunHeuristic} a second time and recursively over each candidate plan \( \pi' \) that does not improve the local solution \( \pi \) after one local search move \( \pi \rightarrow \pi' \),
but has a small enough cost to be a good candidate to start a subsequent meta-heuristic search.

Based on this heuristic, we propose a Heuristic Acceptance Rollout Policy (HARP) in which the initial dispatch plan (\textit{IniPlan}) and the update before dispatch decisions (\textit{DispatchUpdate}) match RP’s functions, but it heuristically solves (7) instead of an IP solver to update the plan upon order arrivals in \textit{ArrivalUpdate}.

7 Computational Experiments

Now we present a series of computational experiments designed over randomly generated instances to test and compare the quality of our heuristic policies. We test all previously discussed policies and add two infeasible solutions to the DDWP-IA operation used as benchmarks: FLEX and LB. The first is a flexible rollout of the \textit{a priori} policy that relaxes immediate request acceptance and postpones it following the DDWP model in [25]; unlike DDWP-IA, it only rejects orders left unserved at the end of the day. LB corresponds to the perfect information lower bound. We run MP with \( d_{\text{max}} = \frac{2T}{T} \), after initial calibration. All policies were programmed in Java and simulated running one thread of a Xeon E5620 processor with up to 12Gb RAM, and using CPLEX 12.6 when necessary as an IP solver.

7.1 Design of data sets for base experiment

We generated 135 data sets, each with a specific geography setting of 50 customer nodes in subset \( I \), a subset \( I_0 \subset I \) of previously accepted commitments, and a vector of request arrival rates \( \lambda \in \mathbb{R}^{|I|} \) for 50 independent Poisson arrival processes. We designed five geography scenarios \( g \in \{0, \ldots, 4\} \), each having 50 different randomly assigned locations following a uniform discrete distribution over a square region of side 50 units; the depot is located at the center of the square region in coordinate (25, 25); we ruled out repeated location coordinates. As in [25], travel times are computed as the \( \ell_1 \)-norm between two locations’ coordinates, and we assume equal values for each edge’s travel time, cost and distance. All data sets share a common continuous time horizon with \( T = 882 \) time units, built to have \( W = 7 \) possible dispatch waves homogeneously distributed over time such that \( t_w - t_{w-1} = 126 \). We also assume a service time at nodes equal to \( u_i = 6 \), and a depot setup time equal to \( u_0 = 20 \). Under this setting, any single-location visit can be served in a single trip taking no more time than a single dispatch wave. The request cut-off time is set at \( 2/7 \) of the horizon, \( i.e., t_{ct} = t_2 = 252 \), and each order processing time is set equal to \( p = 20 \) units. We use a penalty cost of the form \( \beta_i = 2d_{0,i} + 1 \) that is no cheaper than any round-trip covering a single request.
For each geography scenario we set three probabilities $p_{\text{start}} = \{0\%, 15\%, 30\%\}$ to have a pending service at each node $i \in I$ at time $t = T$ and simulated the sets $I^0$ three times $s = \{0, 1, 2\}$ for each value $p_{\text{start}}$. Each data set also has a setting of $\lambda^* \in \{0.5, 1, 2\}$, the expected rate of online requests per node and day, to simulate different levels of request arrival intensity. The arrival rate $\lambda_i$ defining the Poisson process at node $i \in I$ is randomly generated in clusters of 10 nodes so that $10\lambda^* = (T - t^{ct}) \sum_{k=10(k-1)}^{10k} \lambda_i$ for each $k = 1, 2, 3, 4, 5$. We form clusters to create $540 = 4 \cdot 135$ instances with four different network sizes, each one made with the first $n = 20, 30, 40, 50$ nodes of each data set. Each instance is defined by a tuple $(g, p_{\text{start}}, s, \lambda^*)$ : $g \in \{0, 1, 2, 3, 4\}, p_{\text{start}} \in \{0\%, 15\%, 30\%\}, s \in \{0, 1, 2\}, \lambda^* \in \{0.5, 1, 2\}$; all data sets can be found online at sites.google.com/site/maklappor/ddwp-data-sets.

In addition, we simulated $M = 50$ request arrival realizations for each instance to estimate the expected cost of each policy under common random numbers. Table 1 presents all metrics computed for each policy and instance realization, and then averaged for each instance.

### Table 1: Metrics computed for each policy and instance realization

<table>
<thead>
<tr>
<th>metric</th>
<th>definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>cost/request</td>
<td>the total cost divided by the total number of requests (initial &amp; realized)</td>
</tr>
<tr>
<td>fill rate (fr)</td>
<td>the percentage of requests accepted by the vehicle over all realized requests</td>
</tr>
<tr>
<td>travel/order</td>
<td>the policy’s travel time over the total number of orders accepted</td>
</tr>
<tr>
<td>gap$^P$</td>
<td>the percentage increase of the policy’s cost over $P \in {LB, FLEX, RP}$, respectively</td>
</tr>
<tr>
<td>nRoutes</td>
<td>number of vehicle dispatched</td>
</tr>
<tr>
<td>nWaves</td>
<td>average number of waves used by each vehicle route dispatched over the realization</td>
</tr>
<tr>
<td>iWait</td>
<td>number of waves spent waiting at the depot before the initial dispatch</td>
</tr>
<tr>
<td>afterCT</td>
<td>percentage of orders served that are dispatched after the cut-off time</td>
</tr>
<tr>
<td>nodes/route</td>
<td>average number of node visits per route</td>
</tr>
<tr>
<td>time$_{off}$</td>
<td>average off-line initial plan construction time, i.e., a call of $\text{IniPlan}$</td>
</tr>
<tr>
<td>time$_{dis}$</td>
<td>average plan update time before dispatch decisions, i.e., a call of $\text{DispatchUpdate}$</td>
</tr>
<tr>
<td>time$_{acc}$</td>
<td>average plan update time before acceptance decisions, i.e., a call of $\text{ArrivalUpdate}$</td>
</tr>
</tbody>
</table>

7.2 Base experiments

Table 2 presents average results for each policy over all instances. On average, AP and MP have costs 48.0% and 35.0% over the deterministic bound (LB), which may be explained due to a loss of 13.3% and 7.9% in the percentage of requests accepted. Nevertheless, both policies have totally different behavior. MP keeps travel/order low taking advantage of re-optimization capabilities. However, its myopic behavior does not generate enough vehicle returns for recourse possibilities, producing fewer routes, longer route duration,
and more average nodes per route than LB. Unlike MP, the \textit{a priori} policy (AP) becomes inefficient in \textit{travel/order} when expected requests do not realize. However, it creates a plan closer to LB in terms of average number of routes, initial wait at the depot, and average number of waves per route.

Table 2: Average results averaged for each policy

<table>
<thead>
<tr>
<th>metric</th>
<th>LB</th>
<th>FLEX</th>
<th>MP</th>
<th>AP</th>
<th>MPF</th>
<th>RP</th>
<th>HARP</th>
</tr>
</thead>
<tbody>
<tr>
<td>cost/request</td>
<td>11.5</td>
<td>13.3</td>
<td>15.4</td>
<td>17.0</td>
<td>15.2</td>
<td>13.9</td>
<td>14.2</td>
</tr>
<tr>
<td>$fr$</td>
<td>93.4%</td>
<td>88.6%</td>
<td>85.5%</td>
<td>80.1%</td>
<td>85.6%</td>
<td>87.8%</td>
<td>86.9%</td>
</tr>
<tr>
<td>travel/order</td>
<td>9.1</td>
<td>8.8</td>
<td>8.9</td>
<td>9.4</td>
<td>9.0</td>
<td>8.9</td>
<td>8.9</td>
</tr>
<tr>
<td>gap_{LB}</td>
<td>N/A</td>
<td>15.9%</td>
<td>35.0%</td>
<td>48.0%</td>
<td>33.0%</td>
<td>21.0%</td>
<td>23.8%</td>
</tr>
<tr>
<td>gap_{FLEX}</td>
<td>N/A</td>
<td>N/A</td>
<td>16.2%</td>
<td>27.1%</td>
<td>14.5%</td>
<td>4.4%</td>
<td>6.8%</td>
</tr>
<tr>
<td>gap_{RP}</td>
<td>N/A</td>
<td>-4.2%</td>
<td>11.2%</td>
<td>21.8%</td>
<td>9.7%</td>
<td>N/A</td>
<td>2.3%</td>
</tr>
<tr>
<td>$time_{off}$ (sec.)</td>
<td>121.9</td>
<td>529.2</td>
<td>0.00</td>
<td>529.2</td>
<td>529.2</td>
<td>529.2</td>
<td>529.2</td>
</tr>
<tr>
<td>$time_{disp}$ (sec.)</td>
<td>0.00</td>
<td>81.8</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>68.6</td>
<td>80.1</td>
</tr>
<tr>
<td>$time_{acc}$ (sec.)</td>
<td>0.00</td>
<td>0.00</td>
<td>2.4</td>
<td>0.00</td>
<td>1.2</td>
<td>18.1</td>
<td>1.1</td>
</tr>
<tr>
<td>nRoutes</td>
<td>2.69</td>
<td>2.51</td>
<td>1.94</td>
<td>2.48</td>
<td>2.21</td>
<td>2.52</td>
<td>2.52</td>
</tr>
<tr>
<td>iWait</td>
<td>2.87</td>
<td>3.21</td>
<td>3.43</td>
<td>3.09</td>
<td>3.35</td>
<td>3.20</td>
<td>3.21</td>
</tr>
<tr>
<td>nWaves</td>
<td>1.61</td>
<td>1.58</td>
<td>1.86</td>
<td>1.64</td>
<td>1.72</td>
<td>1.59</td>
<td>1.58</td>
</tr>
<tr>
<td>nodes/Route</td>
<td>7.2</td>
<td>7.4</td>
<td>9.2</td>
<td>6.7</td>
<td>8.3</td>
<td>7.4</td>
<td>7.3</td>
</tr>
<tr>
<td>afterCT</td>
<td>68.4%</td>
<td>79.0%</td>
<td>73.3%</td>
<td>78.3%</td>
<td>76.3%</td>
<td>78.7%</td>
<td>79.1%</td>
</tr>
</tbody>
</table>

This suggests combining both policies in MPF, which marginally reduces the cost per request by using AP’s dispatch structure. A more sophisticated blend such as RP produces better results by redesigning the dispatch plan before each decision. Compared to MPF, RP cuts the average cost per request and percentage gap over LB by 8.6% and 36%, respectively. Most benefits arise from improving order acceptance, which is crucial for companies interested in providing the best possible customer service. Refer to Appendix A.4 for a detailed comparative performance of our policies over different instance sizes.

In Figures 6 and 7 we present two important differences between MPF and RP. Experimentally, the latter not only increases request acceptance, but also accepts more requests later in the horizon and further from the depot. Conversely, it seems that MPF concentrates on covering early requests closer to the depot with cheaper insertion costs until it runs out of distribution capacity.

Compared to LB, RP waits longer before the first vehicle dispatch and increases the average percentage of accepted orders dispatched after the cut-off time by 10.3%, pushing its dispatch structure later in time to protect the plan against uncertainty. Compared to FLEX, RP increases $cost/request$ by 4.5% and provides
Figure 6: Average number of orders accepted over each orders’ distance to the depot range.

Figure 7: Average number of orders accepted over the orders’ earliest dispatch wave.

an estimate to managers of the operational cost incurred by imposing immediate order acceptance in SDD.

All policies computing an initial a priori solution share offline computation time (time\textsubscript{off}), but differ in online computation per dispatch (time\textsubscript{disp}) and per request acceptance decision (time\textsubscript{acc}); MP, AP and MPF are simple and fast online policies, while RP requires additional computational effort. HARP provides on average 16.5 times faster request acceptance times, making small sacrifices in cost/request (2.2\% increase); this policy still outperforms both myopic policies, even when these last two use IP solvers to make acceptance decisions. The average solution times time\textsubscript{acc} and time\textsubscript{disp} over all instances aggregated by network size n are displayed in logarithmic scale in Figure 8. As expected due to the nature of exact MIP models, computational times increase exponentially with n. In case of time\textsubscript{acc}, HARP avoids this exponential growth and keeps this average time under 2 seconds for n = 50, verifying that our more sophisticated policies can be implemented in real-time.

In Figure 9 we compare the average cost per request of our best policies, RP and HARP, over all instances sharing parameters of network size n and average arrival rate per node λ∗. We experimentally observe small economies of scale as n grows for instances with low arrival intensity (λ∗ = 0.5), possibly due to order consolidation. Conversely, the cost per order grows with n for moderate and high arrival intensities, indicating congestion in the system; the increased request arrival frequency at nodes may reduce the system’s
marginal acceptance capacity as $n$ grows.

Also, our results suggest that the cost per request is cheaper with higher request frequencies per node for a fixed network size, and this reduction marginally increases as $n$ decreases. This suggests that an instance with fewer nodes and higher requests per node can be managed at lower cost than an instance with a larger network and lower arrival intensity per node, even when both have the same total number of expected requests per day. As Figure 9 shows, instances with 40 nodes and $\lambda^* = 1$ produce an average cost/request 50% higher than instances with $n = 20$ and $\lambda^* = 2$. This result suggests that SDD services distributing to lockers (which consolidate orders) instead of delivering directly to customer homes could be a cheaper option when request density is low.

Figure 10 presents the average cost per order of RP and HARP as a function of the probability of having orders waiting for service before execution starts ($p_{\text{start}}$) and the average arrival intensity ($\lambda^*$). As expected for each graph, the more information available at the start of the operation, the smaller the cost per request.
The cost reduction with $p_{\text{start}}$ is particularly high for low arrival intensity ($\lambda^* = 0.5$); in this case, orders carried over from previous days are relatively more important than online requests; the cost per order tends to stabilize for instances with $\lambda^* = 2$, where off-line orders lose relative importance.

Figure 11 presents the average number of routes and initial wait at the depot ($i_{\text{Wait}}$) for RP as a function of $n$ and $\lambda^*$. For a relatively busier instance (bigger $n$ and $\lambda$), our policy reacts by generating more dispatches and waiting less at the depot; instances with congested and smaller networks ($n = 20, \lambda^* = 2$) produce fewer routes and wait more at the depot than instances with scattered and bigger networks ($n = 40, \lambda^* = 1$).

### 7.3 Analysis of performance sensitivity over processing time

Now we study the sensitivity of RP to the order processing time $p$. We extend our experiments, using all 45 instances with $n = 30$ and $\lambda^* = 1$ and combine them with different values of $p \in \{0, 20, 40, 80, 160\}$ to
generate 225 new instances. We present average results as a function of \( p \) in Figure 12. Our experiments illustrate the direct impact that order processing times have in the performance of an SDD distribution system. Also, an increase in processing time hinders the system in both aspects: routing efficiency, because less dispatch consolidation exists when fewer orders are ready for dispatch at each wave; and request acceptance rate, possibly due to a loss in the system’s overall acceptance capacity.

Additionally, Figure 13 depicts how the average RP gap over LB and gap difference over the myopic policies reduce as \( p \) increases. These results suggest that the cost increase in LB is relatively higher due to a reduction in the feasible space of actions, removing flexibility and complexity. Also, it suggests that the relative value of implementing RP over myopic policies gets reduced as order processing times increase.

### 7.4 Analysis of performance sensitivity under varying levels of information dynamism

We study the performance of RP as a function of the order arrival dynamism related to the cut-off time value \( t^{ct} \). We take again all instances with \( \lambda^* = 1, n = 30 \) and combine with all values of \( t^{ct} \in \{126, 252, 378\} \). To produce comparable instances in terms of number of expected requests, we keep all originally simulated
order arrivals for $t^{ct} = 252$ and distribute them proportionally between $T$ and the new cut-off time (see Appendix A.5). In Table 3 we present average results over all instances as a function of $t^{ct}$. A higher value of $t^{ct}$ indicates orders arriving relatively closer to the start of the horizon (less dynamism), while a lower value of $t^{ct}$ indicates arrivals more dispersed over time (more dynamism). We first observe that the operation becomes cheaper per request with a higher value of $t^{ct}$. An earlier cut-off time produces dispatch decisions executed with more information, and the percentage of accepted orders dispatched after $t^{ct}$ increases from 65% to 94.8%, improving route efficiency and $fr$. The opposite effect is observed with a lower value of $t^{ct}$ and the percentage of orders dispatched after the cut-off drastically reduces to 21.0%. Notably, the average gap over LB decreases with any change from the base setting. The reduction of gap is expected when the cut-off time is earlier, since earlier realized orders render the dynamic policy closer to a deterministic solution; there is equality for the limiting case $t^{ct} = T$. The gap reduction when $t^{ct}$ is later may be related to a reduction in the instance’s acceptance capacities, which adds a fixed cost to both the deterministic and the dynamic solution.

Table 3: Average performance indicators of RP versus cutoff time ($t^{ct}$)

<table>
<thead>
<tr>
<th>$t^{ct}$</th>
<th>cost/request</th>
<th>gapLB(%)</th>
<th>$fr$(%)</th>
<th>travel/order</th>
<th>afterCT(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>126</td>
<td>27.4</td>
<td>21.8</td>
<td>76.2</td>
<td>10.1</td>
<td>21.0</td>
</tr>
<tr>
<td>252</td>
<td>17.1</td>
<td>32.4</td>
<td>86.8</td>
<td>8.9</td>
<td>65.0</td>
</tr>
<tr>
<td>378</td>
<td>9.9</td>
<td>16.0</td>
<td>95.7</td>
<td>7.5</td>
<td>94.8</td>
</tr>
</tbody>
</table>

8 Conclusions

We formulate the DDWP-IA for SDD operations, which integrates immediate request acceptance and processing with order distribution. We design a proactive dynamic policy (RP) that rolls out an optimal *a priori* policy (AP) before each decision epoch visited by the system and compare it to benchmark policies and relaxed policies over a set of computational instances under different settings of geography, problem size, online order arrival intensity, and percentage of accepted orders known before the operation starts. RP outperforms any of our feasibles policies. The success of RP is related to optimization-guided decisions and increasing recourse opportunities by executing more vehicle dispatches compared to other policies. RP especially increases acceptance for orders arriving relatively later in the horizon and farther away from the depot. Compared to a similar policy, which additionally can postpone order acceptance decisions throughout
the day, RP increases its cost per request by 4.5% and provides an estimate to managers of the operational
cost incurred by imposing immediate order acceptance in SDD. To operate our policy in real time, we design
a meta-heuristic to speed up computation incurring a small cost increase (2.1%).

We also conclude that a reduction in the order processing times may be directly transferred to a reduc-
tion in cost per request in the order distribution system suggesting the importance of implementing faster
warehousing operations for SDD.

Future research in the same-day distribution system includes the extension of this model to multiple
vehicles and to study the impact of order dependent service times into a dispatch plan. Another challenge
is to study the strategic allocation of a fixed number of potential dispatch waves over time. It would also be
interesting to allow the system to dispatch over continuous time as [42, 43, 48] for other dynamic routing
problems.

We also see great opportunity in extending the study to one synchronizing warehousing operations (pick-
ing and packaging) with request acceptance and order distribution system. This study considers such inter-
relation only partially via fixed processing times per order. However, one could model economies of scale
in order processing times when two or more orders are picked up and packaged together. Such savings could
impact the order distribution system and our request acceptance capabilities. There are still many open
challenges associated with same-day delivery for the logistics research community.

References


[3] C. Archetti, D. Feillet, and M.G. Speranza, Complexity of routing problems with release dates,

windows and multiple use of vehicles, European Journal of Operational Research 202 (2010), no. 3,
756 – 763.

search 199 (2012), no. 1, 103–112.

[6] ______, An adaptive large neighborhood search for a vehicle routing problem with multiple routes,


A Appendix

A.1 Local Search Neighborhoods

As follows we define three local search (LS) procedures called within our meta-heuristic in Algorithm 2 and based on our heuristic in [25]: IntraLS, InterLS and WavesLS. Each LS procedure explores over different levels of a dispatch plan \( \pi \) structure and searches to improve the best plan available so far \( \pi^* \).

IntraLS, defined in Algorithm 3, exploits the relation between the DDWP and a PC-TSP

\[
PCTSP(d_{\text{max}}, Q, \rho) := \min_{S \subseteq Q: \tau(S) \leq d_{\text{max}}} \left\{ c^*(S) - \sum_{i \in S} \rho_i \right\}
\]

solved over a subset \( Q \subseteq I \) of nodes, prizes \( \rho_i, i \in Q \), and a maximum route duration \( d_{\text{max}} \). IntraLS is a best move procedure, where a move is described by re-optimizing one route \( r^w_{\pi} \) dispatched at wave \( w \) from the local solution \( \pi \). Let \( \pi' \) be a copy of the local solution \( \pi \) after removing route \( r^w_{\pi} \) from it and leaving all remaining routes unaltered. The procedure solves a PC-TSP over the set of nodes in \( \pi' \) left unattended \( \bar{I}(\pi') = \{ i \in I : i \notin r^w_{\pi}, \forall k \in W^\pi \} \), a maximum route duration equal to the duration of the waves left available after removing route \( r^w_{\pi} \), and prizes \( \rho_i = g_{i,0} - g_{i,w} \) defining penalty savings when visiting node \( i \) in a vehicle dispatch at \( w \). The procedure updates the local solution \( \pi \) after each best improvement loop if the best move candidate \( \hat{\pi} \) has a lower cost; it also updates the overall best solution \( \pi^* \). The procedure returns a boolean variable with a true value if the local plan \( \pi \) was improved and returns false if not. Any local solution \( \pi \) processed by IntraLS contains only routes \( r^w_{\pi} \), \( k \in W^\pi \) that are optimally sequenced and that cannot be improved by selecting a different subset of requests to service from \( \bar{I}(\pi) \cup \{ r^w_{\pi} \} \).

Algorithm 3 Intra-route LS procedure

1: \textbf{procedure} INTRA\_LS(plan \( \pi \), best plan \( \pi^* \))
2: \hspace{1em} \( \mu \leftarrow \text{false} \) \hfill //initialize best candidate
3: \hspace{1em} \textbf{loop}
4: \hspace{2em} \( \hat{\pi} \leftarrow \pi \) \hfill //initialize best candidate
5: \hspace{2em} \textbf{for} \( w \in W^\pi \) \textbf{do}
6: \hspace{3em} Let \( \pi' \) be a copy of \( \pi \) without route \( r^w_{\pi} \)
7: \hspace{3em} Let \( d_{\text{max}} \leftarrow t_{w} - t_{q(r^w_{\pi})} \)
8: \hspace{3em} Solve PCTSP\( (d_{\text{max}}, \bar{I}(\pi'), \{ g_{i,0} - g_{i,w} \}) \) and add the optimal route found to \( \pi' \) and dispatch it at wave \( w \)
9: \hspace{3em} \textbf{if} \( (c_{\hat{\pi}} < c_{\pi'}) \) \textbf{then} \( \hat{\pi} \leftarrow \pi' \) \hfill //update best candidate
10: \hspace{3em} \textbf{if} \( (c_{\hat{\pi}} < c_{\pi}) \) \textbf{then}
11: \hspace{4em} \( \pi \leftarrow \hat{\pi} \), \( \mu \leftarrow \text{true} \) \hfill //update local solution
12: \hspace{4em} \textbf{if} \( (c_{\hat{\pi}} < c_{\pi^*}) \) \textbf{then} \( \pi^* \leftarrow \hat{\pi} \) \hfill //update best solution
13: \hspace{2em} \textbf{else} \textbf{break loop}
14: \hspace{1em} \textbf{return} \( \mu \)

InterLS uses best move searches over pairs of routes using neighborhoods inspired by those in [37] for the capacitated vehicle routing problem (CVRP): two-edge exchanges between routes, removal and reinsertion of a \( k \)-customer sequence from one route to another, and customer swaps between routes. To implement these ideas, we account for two differences between the CVRP and the DDWP. First, we model
the prize-collecting component; a move changes penalty savings (due to the different dispatch time). Second, we check the durations of the new routes to ensure that they remain compatible with the fixed dispatch times of the unchanged routes. Just as IntraLS, this function updates the local solution \( \pi \), the best solution \( \pi^* \) and returns \( true \) if the local solution \( \pi \) was updated and \( false \), otherwise.

The third neighborhood search is a Waves Local Search (WavesLS), described in Algorithm 4. The search perturbs the dispatch structure \( \mathcal{W}^{\pi} \) of a plan \( \pi \) using seven operations: Reorder, Cut, Merge, Insert, Delete, Enlarge, and Reduce. The Reorder operator is defined in [25] and uses a job scheduling approach to re-assign the routes dispatched in \( \pi \) to the best possible dispatch waves, without altering the customer visit sequences or the route durations. The last six search over new candidate solutions by changing the dispatch structure of \( \pi \) and solving multiple PC-TSPs.

**Algorithm 4 Waves Local Search (WavesLS)**

1: **procedure** WAVESLS(local plan \( \pi \), best plan \( \pi^* \))
2: **loop**
3: if \( \neg \text{REORDER}(\pi, \pi^*) \) and \( \neg \text{CUT}(\pi, \pi^*) \) and \( \neg \text{MERGE}(\pi, \pi^*) \) and \( \neg \text{INSERT}(\pi, \pi^*) \) and \( \neg \text{DELETE}(\pi, \pi^*) \) and \( \neg \text{ENLARGE}(\pi, \pi^*) \) and \( \neg \text{REDUCE}(\pi, \pi^*) \) then
4: break loop

The Cut operator, described in Algorithm 5, searches over dispatch plans that result when splitting a single vehicle route \( r^w_\pi \) with duration \( w - q(w, r^w_\pi) \geq 2 \) waves into two dispatches with shorter wave duration; this operator adds an extra return trip to the depot, as depicted in Figure 14. The Merge operator, described in Algorithm 6, works reversing cut moves and searches over all dispatch profiles that arise when merging two consecutive dispatches into a single longer duration dispatch, as shown in Figure 15. The Insert operator, described in Algorithm 7, inserts a new route with duration one wave between two dispatched routes in a plan \( \pi \) shifting all previous dispatches a wave earlier in time; see Figure 16. The Delete operator, described in Algorithm 8, searches for a better solution by deleting one dispatch from the plan and pushing all preceding dispatches later in time, as depicted in Figure 17. The Enlarge operator, described in Algorithm 9, searches for a better solution by extending the duration of a vehicle dispatch by one wave and dispatching all preceding routes one wave earlier, as depicted in Figure 18. Finally, the Reduce operator, described in Algorithm 10, searches for a better solution by reducing a wave the duration of one dispatch in the plan and

![Figure 14: Example of a cut operation where a new dispatch plan is created (dashed flow) from an existing one (continuous flow) by adding an extra return to the depot.](image)
Algorithm 5 Cut operation

1: procedure CUT(local plan $\pi$, best plan so far $\pi^*$)
2:   for $w \in \nabla / \pi$ do
3:       $d_{\text{max}} \leftarrow t_w - t_{q(w,r_w^*)}$
4:       for $v : (w-1) \to (w - q(w,r_w^*) + 1)$ do
5:           Let $\pi'$ a copy of $\pi$ without route $r_w^*$
6:           Solve PCTSP($t_w - t_v, \bar{I}(\pi'), \{g_{i,0} - g_{i,w}\}$) and add optimal route to $\pi'$ at wave $w$
7:       end for
8:       if ($c_{\pi'} < c_{\pi}$) then
9:           if ($c_{\pi'} < c_{\pi^*}$) then $\pi^* \leftarrow \pi'$
10:          $\pi \leftarrow \pi'$ and return true
11:     end if
12: end for
13: return false

Figure 15: Example of a merge operation where a new dispatch plan is created (dashed flow) from an existing one (continuous flow) by removing one return to the depot and merging two dispatches.

Algorithm 6 Merge operation

1: procedure MERGE(local plan $\pi$, best plan $\pi^*$)
2:   for $w \in \nabla / \pi$ such that $q(w,r_w^*) > 0$ do
3:     Let $w' \leftarrow q(w,r_w^*)$
4:     Let $\pi'$ be a copy of $\pi$ without routes $r_w^*$ and $r_{w'}^*$
5:     Let $d_{\text{max}} \leftarrow t_w - t_{q(w',r_{w'}^*)}$
6:     Solve PCTSP($d_{\text{max}}, \bar{I}(\pi'), \{g_{i,0} - g_{i,w}\}$) and add optimal route to $\pi'$ at wave $w$
7:     if ($c_{\pi'} < c_{\pi}$) then
8:       if ($c_{\pi'} < c_{\pi^*}$) then $\pi^* \leftarrow \pi'$
9:       $\pi \leftarrow \pi'$ and return true
10:  end if
11: end for
12: return false

Figure 16: Example of an Insert operation where an new dispatch is inserted launching previous routes a wave earlier (dashed flow).
executing all precedent dispatches one wave later, see Figure 19.

**Algorithm 7** Insert operation

1: **procedure** INSERT(local plan $\pi$, best plan $\pi^*$)
2:     if (max{$k \in W^\pi$} = W) then return false
3:     for $w \in W^\pi \cup \{0\}$ do
4:         Let $\pi'$ a copy of $\pi$ with routes $r_\pi^k, k \in W^\pi : k > w$ dispatched one wave earlier.
5:         Solve PCTSP($t_{w+1} - t_w, \bar{I}($$\pi'$), $\{g_i,0 - g_{i,w+1}\}$) and add optimal route to $\pi'$ at wave $w + 1$.
6:         if ($c_{\pi'} < c_\pi$) then
7:             if ($c_{\pi'} < c_{\pi^*}$) then $\pi^* \leftarrow \pi'$
8:             $\pi \leftarrow \pi'$ and return true
9:     return false

**Algorithm 8** Delete operation

1: **procedure** DELETE(local plan $\pi$, best plan $\pi^*$)
2:     for $w \in W^\pi$ do
3:         Let $\pi'$ a copy of $\pi$ with $r_\pi^w$ deleted and all routes $r_\pi^k, k \in W^\pi : k > w$ dispatched $w - q(w,r_\pi^w)$ waves forward in time.
4:         if ($c_{\pi'} < c_\pi$) then
5:             if ($c_{\pi'} < c_{\pi^*}$) then $\pi^* \leftarrow \pi'$
6:             $\pi \leftarrow \pi'$ and return true
7:     return false

A.2 Random Destruction

The heuristic presented in Algorithm 2 calls the function RandomDestruction, defined in Algorithm 11, that partially destroys a solution $\pi$ to move the search to distant solutions and avoid local optimality. This
Figure 18: Example of an Enlarge operation where the last dispatch is enlarged and launched a wave earlier pushing the previous dispatch earlier too (dashed flow).

Algorithm 9 Enlarge operation

```plaintext
1: procedure ENLARGE(local plan \( \pi \), best plan \( \pi^* \))
2:    if (\( \max \{ k \in \mathcal{W}^{\pi} \} = \mathcal{W} \)) then return false
3:    for \( w \in \mathcal{W}^{\pi} \) do
4:        Let \( \pi' \) a copy of \( \pi \) without route \( r^w_{\pi} \) and all routes \( r_k^{\pi}, k \in \mathcal{W}^{\pi} : k > w \) dispatched one wave earlier.
5:        Solve PCTSP(\( t_{w+1} - q(w, r^w_{\pi}), \bar{f}(\pi'), \{ g_{i,0} - g_{i,w+1} \} \)) and add optimal route to \( \pi' \) at wave \( w + 1 \).
6:        if (\( c_{\pi'} < c_{\pi} \)) then
7:            if (\( c_{\pi'} < c_{\pi^*} \)) then \( \pi^* \leftarrow \pi' \)
8:                \( \pi \leftarrow \pi' \) and return true
9:        return false
```

Figure 19: Example of a Reduce operation where one dispatch is reduced and launched one wave later pushing the previous dispatch later too (dashed flow).

Algorithm 10 Reduce operation

```plaintext
1: procedure REDUCE(local plan \( \pi \), best plan \( \pi^* \))
2:    for \( w \in \mathcal{W}^{\pi} \) such that \( w - q(w, r^w_{\pi}) > 1 \) do
3:        Let \( \pi' \) a copy of \( \pi \) without route \( r^w_{\pi} \) and all routes \( r_k^{\pi}, k \in \mathcal{W}^{\pi} : k > w \) dispatched one wave later.
4:        Solve PCTSP(\( t_{w-1} - q(w, r^w_{\pi}), \bar{f}(\pi'), \{ g_{i,0} - g_{i,w-1} \} \)) and add optimal route to \( \pi' \) at wave \( w - 1 \).
5:        if (\( c_{\pi'} < c_{\pi} \)) then
6:            if (\( c_{\pi'} < c_{\pi^*} \)) then \( \pi^* \leftarrow \pi' \)
7:                \( \pi \leftarrow \pi' \) and return true
8:        return false
```
function works at two levels of a solution’s structure and deletes randomly chosen routes and customers from it.

**Algorithm 11** Random destruction procedure for plan $\pi$

1: procedure RANDOMDESTRUCTION(local solution $\pi$)
2: \hspace{1em} Generate a random number $x \in \{1, \ldots, \lfloor 0.5|W_\pi| \rfloor \}$
3: \hspace{1em} Randomly delete $x$ routes from plan $\pi$.
4: \hspace{1em} if ($\pi$ is empty) then return
5: \hspace{2em} for ($w \in W_\pi$) do
6: \hspace{3em} Set $y$ equal to the number of nodes visited in $r^\pi_w$.
7: \hspace{3em} if ($y > 1$) then
8: \hspace{4em} Generate a random number $z \in \lceil 0.25y \rceil, \ldots, \lfloor 0.75y \rfloor$.
9: \hspace{4em} Randomly delete and skip $z$ visits from $r^\pi_w$
10: return

### A.3 Heuristic Solution For the prize-collecting TSP

In Algorithm 12 we implement a metaheuristic solution to a PC-TSP over the set of nodes $Q$, prizes $p_i, i \in Q$, and maximum duration $d^{max}$. This solution implements simulated annealing running over an elementary route $r = \{0, i_1, i_2, \ldots, 0\}$ with objective value $v_p$ with a subset $Q'_{out} \subseteq Q$ of unattended nodes.

The parameters $T_0$ and $\delta$ control the evolution of simulated annealing and $k^{max}$ determines a maximum number of iterations. The search also executes a partial solution destruction when no improvement is found to avoid local optimality. The neighborhood $\mathcal{N}(r)$ used in line 10 is a compound one consisting of ten polynomially sized neighborhoods, each one based on the following moves:

1. a swap between an unvisited node $i \in Q'_{out}$ and a visited node $j$ in route $r$ ($\mathcal{O}(|Q|^2)$ moves),
2. an insertion of an unvisited node $i \in Q'_{out}$ after a visited node $j$ in route $r$ ($\mathcal{O}(|Q|^2)$ moves),
3. a removal of a visited node $j$ from route $r$ ($\mathcal{O}(|Q|)$ moves),
4. a removal of a visited node $k$ from route $r$ and the insertion of an unvisited node $i \in Q'_{out}$ after a visited node $j$ in route $r$ ($\mathcal{O}(|Q|^3)$ moves),
5. 2-opt, i.e., 2-edge exchanges within route $r$ ($\mathcal{O}(|Q|^2)$ moves),
6. all possible removal of a series of $k$ nodes in $r$ starting with node $i$ and its reinsertion after node $j$ in $r$ () after $r_1, \ldots, r_{i+k}$ after $r_j$ ($\mathcal{O}(|Q|^3)$ moves),
7. a internal swap in route $r$ between two visited nodes ($\mathcal{O}(|Q|^2)$ moves),
8. a swap between one visited node $i$ in route $r$ an two nodes $j, k \in Q_{out}$ inserted in series ($\mathcal{O}(|Q|^3)$ moves),
9. a swap between two consecutive visited nodes $i, j$ in route $r$ an one node $k \in Q_{out}$ ($\mathcal{O}(|Q|^2)$ moves),
Algorithm 12 Metaheuristic for the PC-TSP

1: procedure HPCTSP($d_{\text{max}}, Q, \rho, k^\text{max}$),
2: Initialize best route: $r^* \leftarrow \emptyset$,
3: for $s = 0$ to NumSeeds do
4:   Set random seed $s$,
5:   Generate route $r$ by sequentially inserting random nodes from $Q$ into the last position of $r$; stop before violating the route duration constraint,
6:   Update the set of non-visited nodes $Q_{\text{out}}' \leftarrow Q$,
7: Initialize temperature $T \leftarrow T_0$ and iteration counter $k \leftarrow 0$.
8: while ($k < k^\text{max}$) do
9:   update $\leftarrow$ false
10:   for (neighbor $r' \in \mathcal{N}(r)$) do
11:      Generate a random number $p$ for a $\text{Uniform}(0, 1)$ distribution,
12:      if ($e^{(v_{r'} - v_r)/T} > p$) then
13:         $r \leftarrow r'$, update $Q_{\text{out}}'$, update $\leftarrow$ true, and break for.
14:   if ($\neg$update) then Randomly skip and remove 30% of nodes from $r$. Update $Q_{\text{out}}$.
15:      $k \leftarrow k + 1$.
16:      $T \leftarrow T \times \varepsilon$.
17:      if ($v_r > v_{r^*}$) then
18:         $r^* \leftarrow r$,
19:      reset search $T \leftarrow T_0$, $k \leftarrow 0$.
20: return $r^*$
A swap between two consecutive visited nodes $i, j$ in route $r$ and two nodes $k, q \in Q_{out}$ inserted in series ($O(|Q|^3)$ moves).

All moves resulting in violations of the maximum duration limit are discarded.

### A.4 Average heuristic performance over instance size

In Figure 20 we show the performance of RP and MPF in terms of order fill rate ($fr$) and distance/order for all instances having the same network size $n$; we also include the two benchmarks LB and FLEX. In all cases, $fr$ decreases as $n$ increases and RP improves its request fill rate gain over MPF as $n$ increases. All policies are comparable in route efficiency (travel/order), which decreases with the network size (due to increased consolidation opportunities).

### A.5 Generation of comparables instances with different values of cut-off time

To produce comparable instances in terms of number of expected requests, we keep all originally simulated order arrivals for $t^{ct} = 252$ and distribute them proportionally between $T$ and the new cut-off time. Formally, an arrival at $x_0 \in [t^{ct}_0, T]$ is relocated to time $x_1 = t^{ct}_1 + \frac{T - t^{ct}_0}{T - t^{ct}_1}(x_0 - t^{ct}_0) \in [t^{ct}_1, T]$ when varying the cutoff time from $t^{ct}_0$ to $t^{ct}_1$. Figure 21 presents an example with three requests relocated when $t^{ct}$ changes from 252 to 126 and to 378.
Figure 21: Example of request arrivals re-scaled from $t_{ct} = 252$ to $t_{ct} = 126$ and to $t_{ct} = 378$