Request Acceptance in Same-Day Delivery

Mathias A. Klapp[1], Alan L. Erera[2], Alejandro Toriello[2]

[1]Engineering School, Pontificia Universidad Católica de Chile
Santiago, Chile
maklapp *at* ing.puc *dot* cl

[2]H. Milton Stewart School of Industrial and Systems Engieering, Georgia Institute of Technology
Atlanta, Georgia 30332
{aerera, atoriello} *at* isye *dot* gatech *dot* edu

June 2, 2020

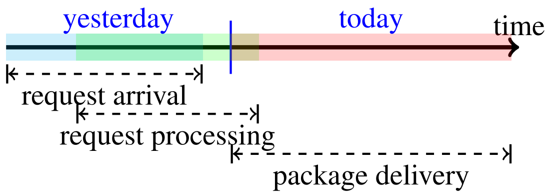
Abstract

We study request acceptance dynamics in same-day delivery systems by formulating the Dynamic Dispatch Waves Problem with Immediate Acceptance, which models integrated management and distribution for dynamically arriving customer requests. We consider an e-commerce platform that attempts to serve most customers with same-day delivery service. When a shopper attempts to place an order, a decision is made immediately to offer same-day delivery service (accept the request) or to deny service (with a penalty cost). Accepted requests are not available for immediate dispatch; they must be processed (picked and packed) before they are loaded for delivery. Vehicle routes are updated dynamically and serve each accepted delivery request no later than the end of the service day. In this work, we limit the study to the case of a single vehicle serving requests, potentially using multiple trips from the distribution center. The objective is to make request acceptance and distribution decisions that minimize the expected sum of vehicle travel costs and penalties for service denials. 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 *a priori* plan and for updating the plan using a heuristic roll-out procedure. Our methods are tested on a family of simulated instances against two common-sense benchmarks and an infeasible relaxed policy that allows the dispatcher to delay the acceptance or rejection of a request until the end of the service day. We demonstrate in our computational study that the cost-per-request of the best benchmark policy is on average 9.7% higher than our proposed dynamic policy and furthermore that the dynamic policy leads to only a 4.4% cost increase over the infeasible relaxed policy lower bound.

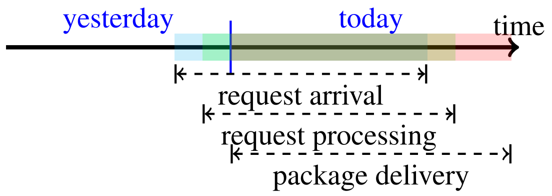
1 Introduction

Same-day delivery (SDD) is a last-mile distribution service offered by retailers to customers who make online purchases and expect fast fulfillment of requested products. It has been implemented by e-commerce companies to enhance customer satisfaction, *e.g.*, Amazon, Instacart, Walmart and Google. Today more than 77 million U.S. residents live in ZIP codes where Amazon offers SDD [23]. Same-day delivery has in part 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 component of the e-commerce 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, increasing competitiveness within the e-commerce sector forces service providers to continuously search for last-mile logistics cost reductions.

In contrast to traditional request fulfillment and delivery systems, in SDD the service provider must operate a system where request arrivals, acceptance, 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 requests 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, 49] for broader discussions on SDD challenges.



(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 customer delivery policy in a model where geographically located delivery requests realize according to some stochastic process over time. It assumes that requests 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, the vehicle can wait for the next wave to potentially accumulate and load more delivery requests or it 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 the decision to accept a customer request is not finalized until the order is loaded for dispatch and that the decision to deny a request is not finalized until the end of the operating day. The DDWP model is most applicable when an outsourcing option is available, and the cost of outsourcing an order (*e.g.*, to an on-demand courier) is modeled as the penalty cost for denying a request. If an outsourcing option is not available, then this assumption means that some customers will place orders that are denied service later in the operating day (usually with a deferral to the following day) and the penalty represents the loss of customer goodwill that results from this action. When a reliable and reasonably-priced outsourcing service is not available in a same-day delivery system, it would be better to use a decision framework that informs customers when shopping or when finalizing an order in the "checkout" process whether they can receive same-day delivery service. Our goal is to propose such a framework. In this paper, if a customer is offered same-day service for a request, we label it as *accepted* and it must be delivered today. A request that is *rejected* will refer to one that is not offered same-day delivery service when shopping or at checkout; typically, these customers will be offered delivery on a later operating day.

This paper presents a new model for this scenario referred to as the Dynamic Dispatch Waves Problem with Immediate Acceptance (DDWP-IA). Compared to the DDWP framework which can delay the rejection (outsourcing) decision for any request until later in the operating day, the DDWP-IA framework immediately decides which order requests to serve and is more realistic when no reasonable outsourcing option exists. Solutions that result from the DDWP-IA framework have higher unit delivery costs for orders served than DDWP solutions simply because there is less flexibility to modify delivery operations later in the operating day, but we show via a computational study that this cost increase is likely to be modest.

In a typical DDWP-IA setting, an e-commerce customer has completed a shopping cart, provided a delivery address, and is shown delivery options. If this customer request can be accommodated for same-day delivery, this option will be available to the customer and, if selected, the request is considered accepted. Otherwise, the customer is not shown the same-day delivery option and only next-day (or longer) delivery times are presented; in this case, the request is considered rejected. To cite one example, the Amazon Fresh service currently operates this way in the U.S.

The DDWP-IA can also be applied when browsing e-commerce customers know which items are available for same-day delivery even before they are placed in a shopping cart. This application is a bit trickier because the customer may take some time to complete a shopping cart and begin the check-out process. During this remaining shopping time, it is possible that same-day delivery service is no longer feasible (or cost-practical) to offer to the customer.

1.1 Contribution

We formulate the DDWP-IA as a semi-Markov decision process (SMDP) [32], which integrates immediate request acceptance with package distribution decisions for SDD systems with a single vehicle. The 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 requests are accepted earlier in the operating day, but we incur additional travel cost and reduce the flexibility of the dispatch system to accommodate requests that appear later. Conversely, if we leave delivery resources available for the future, we may reject too many requests early in the day which may result in an under-utilized vehicle. The DDWP-IA also includes a request acceptance framework that seeks routing economies created by geographic consolidation with previously accepted requests, *e.g.*, accepting a request with a delivery location close to others already waiting for service may only lead to a small marginal increase in travel cost and vehicle travel time. We propose policies that proactively search for such geographic consolidation opportunities, meaning we consider where each planned delivery is located both for accepted requests and also future request delivery location realizations.

In this research, we develop dynamic policies for the DDWP-IA, where a system state is coupled with a state-feasible vehicle dispatch plan (that includes potentially multiple trips from the distribution center for the single vehicle) serving all requests previously accepted and not yet served along with a set of potential future delivery requests that have not yet realized. We show that effective dynamic policies can be constructed by a heuristic roll-out of an optimal *a priori* dispatch plan, which specifies most decisions in advance and allows only simple plan corrections (recourse rules). The dispatch plan is used to guide both vehicle dispatch decisions and request acceptance decisions. This modeling approach is similar to the "Route-based MDP" 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 proposed dynamic policy in a computational study and compare with two simpler heuristics: a myopic re-optimization policy that ignores potential future arrivals when making decisions and uses only information about previously accepted requests, and a policy that fixes dispatch waves according to an initial *a priori* solution, but dynamically accepts and assigns requests 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 lower bound [10, 35] and to the DDWP in [25], which can delay request acceptance decisions. When compared with the DDWP, the study reveals a 4.4% cost increase when imposing immediate request acceptance on the SDD test instances on average.

Finally, we remove the assumption of negligible request 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. The computational study also provides an estimate of the negative impact of processing times on the performance of SDD systems.

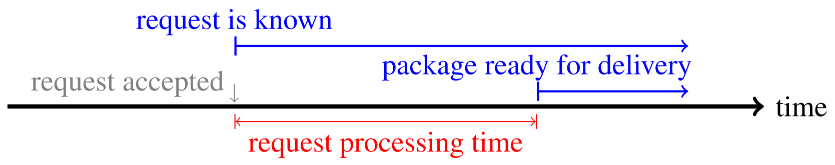


Figure 2: Illustration of request disclosure and ready times.

The remainder of this paper 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 request 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 customer requests realize with a given probability after an initial solution is planned. The works in [12, 16, 24, 27, 48] 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, 46, 43, 50] 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 distribution component and were primarily conceived for request 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 request arrivals and the dispatcher can dynamically insert new customer visits enroute. In contrast, the DDWP-IA includes request acceptance and 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 [47], The Dynamic Multiperiod Routing Problem (DMPRP) [2, 44, 45], The Same Day Delivery Problem for Online Purchases [46, 49] 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 VRPs, these problems typically have vehicles execute multiple trips per day [4, 6]. Also, some explicitly model package release times 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 delivery decisions must be executed simultaneously in the same day. In SDD, it may be inefficient (and potentially infeasible) to wait until all accepted requests are known before executing the delivery plan. An effective operation may execute multiple vehicle dispatches throughout the day deciding the duration of each dispatch with partial demand information. This makes online and *a priori* route planning for SDD significantly more complex than for next-day delivery.

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 offiline simulation and asumed 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 requests to be picked up at a depot and delivered to the customer's location. This difference implies specific package release times at the depot determining the earliest possible time to load the requested packege into a vehicle. Also, the DDWP-IA works in continuous time and makes immediate request 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 requests 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 request's arrival time; *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 request arrival times at the depot. To execute request acceptance decisions, a scenario-based planning approach similar to [8] is used to heuristically estimate a curstomer service insertion profit over multiple simulated future demand 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, 45]. Compared to this pricing-based mechanism, our model assumes a given stochastic demand for service, in which the decision maker directly chooses which customer requests to accept; it is simpler, and it 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 γd_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 $\mathscr{T} = [T, 0]$; 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 accepted requests, 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 $\mathscr{W} := \{W, \ldots, 1\}$ be the discrete set of waves, *i.e.*, feasible vehicle dispatch points, such that each wave $w \in \mathscr{W}$ occurs at time $t_w \in \mathscr{T}$ 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 $\mathscr{W}(t) := \{w \in \mathscr{W} : t_w \leq t\}$; similarly let $\mathscr{W}_0(t) := \mathscr{W}(t) \cup \{0\}$.

Customer delivery requests arrive over time according to a stochastic counting process $N_i(t) \in \mathbb{Z}_+, t \in \mathscr{T}$, 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 request arrival rate λ_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 request 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 $w \in \mathscr{W}(t - p)$. Any rejected delivery request to location i is lost at a penalty cost $\beta_i > 0$.

Dispatch decisions may occur when the vehicle is available at the depot at times t_w, for some wave $w \in \mathscr{W}$. A vehicle dispatch executing a delivery route $r = \{0, i_1^r, \ldots, i_{m_r}^r, 0\}$ with m_r location visits has a transportation cost $\gamma d(r) := \gamma \sum_{j=1}^{m_r+1} \{d_{(i_{j-1}^r, i_j^r)}\}$ and spends $t(r) := d(r) + \sum_{j=0}^{m_r} u_{i_j^r}$ 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 $q(w,r) := \max\{k \in \mathscr{W}_0(t_w - t(r))\}$. 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 physical space capacity of the vehicle is not constraining in this problem, since we assume that customer requests are for small parcels.

The system's state is described at any time $t \in \mathscr{T}$ by $s = (t, \mathbf{a}, w) \in \mathscr{S}$, where $\mathscr{S}$ is the set of all possible system's states. The parameter $\mathbf{a}$ is the vector of open commitments, and each component indexed by $i \in I$ indicates the earliest wave $a_i \in \mathscr{W}$ in which a vehicle dispatch visiting i can cover all its pending accepted requests. There are no pending visits to i when $a_i = \infty$. Given $\mathbf{a}$, let $I_{\mathbf{a}} := \{i \in I : a_i < \infty\}$ be the subset of nodes with pending services. The parameter $w \in \mathscr{W}_0(t)$ 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 requests at i because all the requests' associated services can be executed in one visit dispatched no earlier than $\min(w, a_i)$ without loss of optimality.

A state s is possible if there exists a dispatch plan that feasibly serves all commitments in $\mathbf{a}$. A dispatch plan $\pi = \{r_k^\pi : k \in \mathscr{W}^\pi\}$ is defined by a set of elementary routes r_k^π indexed by its set of dispatch waves $\mathscr{W}^\pi \subset \mathscr{W}$. Formally, for $t \in \mathscr{T}, w \in \mathscr{W}_0(t)$, a state $s = (t, \mathbf{a}, w)$ is possible if and only if there exists a feasible dispatch plan π satisfying:

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

2. Plan π covers all commitments, *i.e.*, if $i \in I_{\mathbf{a}}$, then there exists $k \in \mathscr{W}^\pi$ such that $k \leq a_i$ and $i \in r_k^\pi$.

3. Routes in π do not overlap in time, *i.e.*, for any two consecutive dispatches $k^+ > k^-$ in $\mathscr{W}^\pi$ we have that the next available wave with the vehicle available at the depot after k^+ occurs no later that k^-: $q(k^+, r_{k^+}^\pi) \geq k^-$.

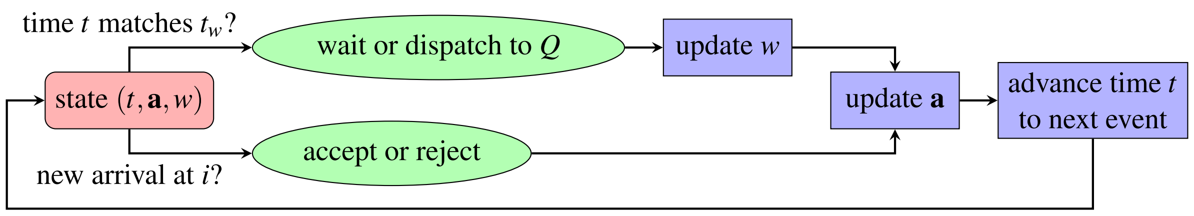


Figure 3: Flowchart of system transitions and actions in state (t,a,w) for the DDWP-IA

A plan's feasibility condition only depends on $\mathbf{a}$ and w and is preserved over time until the earliest possible dispatch wave w; if π is feasible for state $(t,\mathbf{a},w)$, then it is also feasible for any state $(t',\mathbf{a},w) : t' \in [t_w, t]$. 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 services are released for dispatch on or before wave w and any possible state s has a feasible plan π consisting of a single vehicle route dispatched after w covering all commitments.

3.1 Actions, transitions and costs

An accept/reject decision is immediately taken after a request realizes at location i in state $s = (t,\mathbf{a},w)$. Rejecting a request costs β_i, but keeps the system's state unaltered. Accepting it is free of charge, but can only be performed if the post-decision state $(t,\mathbf{a}^i,w)$ remains feasible, where $\mathbf{a}^i$ is the updated vector of commitments. Denote the request's earliest dispatch wave from the depot as $b := \max\{k \in \mathscr{W}(t-p)\}$; then $\mathbf{a}^i$ 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 $(t_w,\mathbf{a},w)$. 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 $(t,\mathbf{a}(w,Q),q^*(w,Q))$, where $\mathbf{a}(w,Q)$ is the updated vector of commitments defined as $a(w,Q)_i = a_i$ when $i \notin Q$ and ∞ 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 $(t,\mathbf{a},w)$.

3.2 Dynamic programming model

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

$$\phi(i,t',t) := \mathbb{P}(\{\tau_i = t'\} \bigcap \{\tau_i > \tau_j, \forall j \neq i\} | \tau_i < t)$$

be the probability density of the next request arriving at time $t' \in [t, t^{ct}]$ at location $i \in I$, and let

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

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

$$C(0, \infty, 0) = 0, \tag{1a}$$

$$C(t_w, \mathbf{a}, w) = \min_{Q \subseteq I : (t, \mathbf{a}(w,Q), q^*(w,Q)) \in \mathscr{S}} \{\gamma d^*(Q) + C(t_w, \mathbf{a}(w,Q), q^*(w,Q))\}, \qquad \forall (t_w, \mathbf{a}, w) \in \mathscr{S} \tag{1b}$$

$$C(t, \mathbf{a}, w) = \psi(t, t_w) C(t_w, \mathbf{a}, w) + \sum_{i \in I} \int_{t'=t_w}^{t} \phi(i, t', t) \tilde{C}(t', \mathbf{a}, w, i) dt', \qquad \forall (t, \mathbf{a}, w) \in \mathscr{S} : t > t_w, \tag{1c}$$

$$\tilde{C}(t, \mathbf{a}, w, i) = \min\{\beta_i + C(t, \mathbf{a}, w), C(t, \mathbf{a}^i, w) : (t, \mathbf{a}^i, w) \in \mathscr{S}\}, \qquad \forall i \in I, (t, \mathbf{a}, w) \in \mathscr{S} : t > t_w, \tag{1d}$$

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 $(t_w, \mathbf{a}, w)$ is equal to the minimum sum of dispatch cost $\gamma d^*(Q)$ plus the post-decision cost-to-go $C(t_w, \mathbf{a}(w,Q), q^*(w,Q))$ 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 $C(t, \mathbf{a}, w) : t > t_w$ is equal to the cost-to-go in the next dispatch decision $C(t_w, \mathbf{a}, w)$ if no requests arrive between t and t_w (with probability $\psi(t, t_w)$), or equal to $\tilde{C}(t', \mathbf{a}, w, i)$ if the next request occurs at node i and time $t' \in [t_w, t]$ (with probability density $\phi(i, t', t)$); $\tilde{C}(t, \mathbf{a}, w, i)$ represents the cost-to-go immediately before the acceptance decision defined in Equation (1d), equal to the minimum cost-to-go between rejecting the request $C(t, \mathbf{a}, w) + \beta_i$ and accepting it $C(t, \mathbf{a}^i, w)$ (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 $t \in \mathscr{T}$ 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 a request before its release time, meaning that the maximum number of requests 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 requests. Figure 4 provides an example where functions $N_i(t)$ and $n_{i,w}$ are depicted for a particular location i.

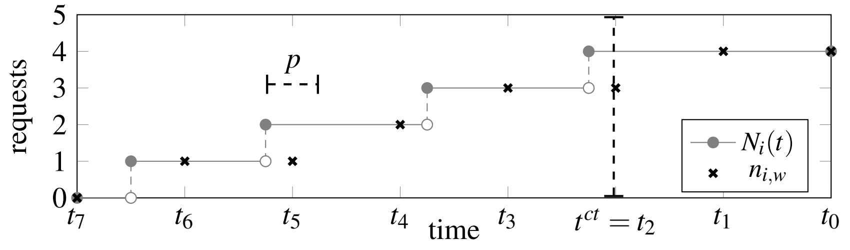


Figure 4: Evolution of requests disclosed ($N_i(t)$) and released ($n_{i,w}$) over time for a particular realization (ω) of arrivals at node i in a seven-wave horizon, a cutoff time $t^{ct} = t_2$, and a processing time p.

Define $h(i,j) = \min\{w \in \mathscr{W} : t_w \geq t_{\{i,j\}} + u_i + u_j + \mathbb{I}_{(i>0,j>0)}(u_0) + \mathbb{I}_{(i>0)}(t_{\{0,i\}}) + \mathbb{I}_{(j>0)}(t_{\{0,j\}})\}$ 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 $w \in \mathscr{W}$ as the sets of feasible nodes and edges for a vehicle dispatch at wave w. Also, define the cut set $E_w(S) = \{\{i,j\} \in E_w : i \in S, j \notin S\}$, for any subset $S \subseteq I_w$; and define $\bar{t}_e = t_e + 0.5u_i + 0.5u_j$ as the adjusted time spent at edge $e = \{i,j\} \in E$ considering service times. The Integer Program in (2) solves the deterministic DDWP,

$$C^D(\mathbf{a}^0, \mathbf{N}(t)) = \min_{\{\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{v},\mathbf{s}\}} \sum_{i \in I} \beta_i \{n_{i,0} z_i + \sum_{w=h(0,i)}^{W} (n_{i,0} - n_{i,w}) y_i^w\} + \sum_{w \in \mathscr{W}} \sum_{e \in E_w} \gamma d_e x_e^w \tag{2a}$$

$$\text{s.t.} \sum_{w=h(i,0)}^{a_i^0} y_i^w = 1, \qquad \forall i \in I_{\mathbf{a}^0} \tag{2b}$$

$$z_i + \sum_{w=h(i,0)}^{W} y_i^w = 1, \qquad \forall i \in I \tag{2c}$$

$$\sum_{e \in E_w(0)} x_e^w \leq 2, \qquad \forall w \in \mathscr{W} \tag{2d}$$

$$\sum_{e \in E_w(S)} x_e^w \geq 2 y_i^w, \qquad \forall w \in \mathscr{W}, \forall S \subseteq I_w, \forall i \in S \tag{2e}$$

$$\sum_{e \in E_w} \bar{t}_e x_e^w \leq \sum_{k<w} (t_w - t_k) v_k^w, \qquad \forall w \in \mathscr{W} \tag{2f}$$

$$\sum_{k<W} s_k + \sum_{k<W} v_k^W = 1, \tag{2g}$$

$$\sum_{k<w} v_k^w = \sum_{k>w} v_w^k + s_w, \qquad \forall w \in \mathscr{W} \setminus \{W\} \tag{2h}$$

$$v_k^w \in \{0,1\}, \qquad \forall w, k \in \mathscr{W}_0 : k < w \tag{2i}$$

$$s_k \in \{0,1\}, \qquad \forall k \in \mathscr{W}_0 : k < W \tag{2j}$$

$$z_i \in \{0,1\}, \qquad \forall i \in I \tag{2k}$$

$$y_i^w \in \{0,1\}, \forall i \in I_w, \text{ and } x_e^w \in \{0,1,2\}, \forall e \in E_w, \qquad \forall w \in \mathscr{W} \tag{2l}$$

where variable z_i is equal to 1 if node i isn't visited, and 0 otherwise; y_i^w is equal to 1 if a dispatch at wave w visits node i, and 0 otherwise; x_e^w is equal to $m \in \{0,1,2\}$ if the vehicle traverses edge e m times at a dispatch in wave w; v_k^w 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 $\mathbf{a}^0$ and (2c) guarantee visiting each node i at most once at wave w. Constraints (2d) - (2e) guarantee that vector $\mathbf{x}^w$ defines a feasible tour only visiting nodes selected by the vector $\mathbf{y}^w$. Constraints (2f) force routes to satisfy duration limits determined by v_k^w. 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 $(\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{v},\mathbf{s})$ representing a feasible vehicle dispatch plan starting after wave W, *i.e.*, satisfying constraints (2c) through (2l), belongs to domain $\mathscr{D}(W)$.

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 π 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 request acceptance and vehicle dispatch decisions, and it is dynamically updated when new information becomes available. Each policy P constructs an initial dispatch plan π that is feasible for the initial state s^0, defined by routes r_k^π dispatched at waves $k \in \mathscr{W}^\pi$, each visiting a subset $Q_k^\pi \subseteq I$ of nodes. After the operation starts, policy P dynamically updates π to keep it feasible throughout all states $s^j \in \mathscr{S}, j = \{0,1,2,3,4,\ldots\}$ 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 $\mathscr{L}$, 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 $C^P(\mathbf{a}^0,\mathscr{L})$ of a policy P given a list of realized requests $\mathscr{L}$ and $\mathbf{a}^0$. P is determined by three functions used to update the plan: **IniPlan**, **ArrivalUpdate** and **DispatchUpdate**.

Algorithm 1 initializes the state of the system in line 2 and calls function **IniPlan**, which constructs an initial plan π 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 π, calling **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 **DispatchUpdate** in line 12 and, if w belongs to the set of planned dispatches, it dispatches route r_w^π 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 **ArrivalUpdate** is critical to allow fast acceptance

Algorithm 1 Implementation of a generic policy P

```
1: procedure ExecutePolicy($P$,$\mathbf{a}^0$,$\mathscr{L}$)
2:     Initialize cost and state: $C \leftarrow 0$, $s := (t, \mathbf{a}, w) \leftarrow (T, \mathbf{a}^0, W)$
3:     Initialize plan: $\pi \leftarrow$IniPlan($P$,$\mathbf{a}^0$)
4:     while (unprocessed requests ($\mathscr{L} \neq \emptyset$) 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 $\mathscr{L}$, get its location $i$ and release wave $b \leftarrow \max\{x \in \mathscr{W}_0(t-p)\}$
8:             Update plan: $\pi \leftarrow$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)
12:            Update plan: $\pi \leftarrow$DispatchUpdate($P, \pi, s$)
13:            if plan $\pi$ dispatches at wave $w$ then
14:                dispatch route $r_w^\pi$: $C \leftarrow C + \gamma t(r_w)$, $\mathbf{a} \leftarrow \mathbf{a}(w, Q_w^\pi), w \leftarrow q(w, r_w^\pi)$, remove $r_w^\pi$ 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 $(t, \mathbf{a}, w)$ 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 **ArrivalUpdate**(MP, π, s, i, b)

$$\min_{\{\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{v},\mathbf{s}\} \in \mathscr{D}(w)} \beta_i \{z_i + \sum_{k=\max(b+1, h(0,i))}^{w} y_i^k\} + \sum_{k=1}^{w} \sum_{e \in E_k} \gamma d_e x_e^k \tag{3a}$$

$$\text{s.t.} \sum_{k=h(0,i)}^{\min\{a_i, w\}} y_i^k = 1, \qquad \forall i \in I_{\mathbf{a}}. \tag{3b}$$

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.*, $\{\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{v}, \mathbf{s}\} \in \mathscr{D}(w)$ and constraints (3b) guarantee the coverage of all previous commitments. For the Myopic policy, **DispatchUpdate**(MP, π, s) does not alter the previous plan π. Finally, function **IniPlan**$(MP, \mathbf{a}^0)$ determines an initial plan π with a single route equal to an optimal TSP route over $I_{\mathbf{a}^0} \cup \{0\}$ 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 requests, 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^{max} to enforce this behaviour in (3). The value of parameter d^{max} must be calibrated beforehand.

5.2 *A priori* policy

Now we present an *a priori* policy (AP) in which a static dispatch plan π 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 λ_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.

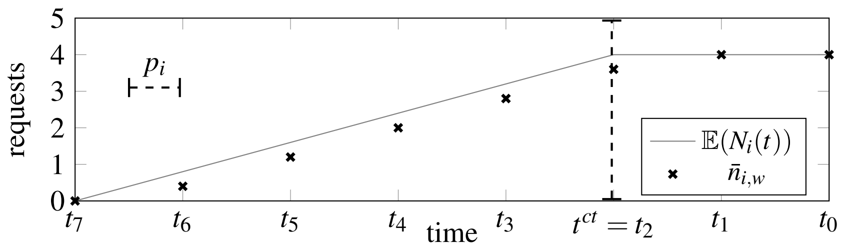


Figure 5: Expected requests arrived at time t ($\mathbb{E}(N_i(t))$) and released for delivery to node i at wave w ($\bar{n}_{i,w}$) for a particular Poisson arrival process with $t^{ct} = t_2$ and $\lambda_i = 0.8$.

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 $\bar{n}_{i,w}$ requests released at any node i and wave w. The IP that solves for this optimal *a priori* plan is

$$\min_{\{\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{v},\mathbf{s}\} \in \mathscr{D}(W)} \sum_{i \in I} \beta_i \{\bar{n}_{i,0} z_i + \sum_{w=h(0,i)}^{W} (\bar{n}_{i,0} - \bar{n}_{i,w}) y_i^w\} + \sum_{w=1}^{W} \sum_{e \in E_w} \gamma d_e x_e^w \tag{4a}$$

$$\text{s.t.} \sum_{w=h(i,0)}^{W} y_i^w = 1, \qquad \forall i \in I_{\mathbf{a}^0}; \tag{4b}$$

it shares its feasible region with (2), but has an objective determined by expected future request rejections. The function **IniPlan**$(AP, \mathbf{a}^0)$ returns a dispatch plan solving (4); function **ArrivalUpdate**(AP, π, s, i, b) produces no change to the plan; and **DispatchUpdate**(AP, π, s) improves the performance of the plan at each wave $w \in \mathscr{W}^{\pi}$ in state s by skipping from the upcoming dispatch all planned visits without pending services 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 $\mathscr{W}^{AP}$ be the waves used by a solution to (4). At any state s visited by the system, MPF keeps a feasible dispatch plan π to s that only plans dispatches at waves in $\mathscr{W}^{AP}$. The plan π is initialized calling function **IniPlan**$(MPF, \mathbf{a}^0)$, with an optimal *a priori* plan. When a request arrives at state $(t, \mathbf{a}, w)$ and node i with release wave b the plan calls function **ArrivalUpdate**(MPF, π, s, i, b), which solves

$$\min_{\{\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{v},\mathbf{s}\} \in \mathscr{D}(w)} \beta_i \{z_i + \textstyle\sum_{k=\max(b+1,h(0,i))}^{w} y_i^k\} + \sum_{k=1}^{w} \sum_{e \in E_k} \gamma d_e x_e^k \tag{5a}$$

$$\text{s.t.} \sum_{k=h(0,i)}^{\min\{a_i, w\}} y_i^k = 1, \qquad \forall i \in I_{\mathbf{a}}, \tag{5b}$$

$$v_k^w = 0, \qquad \forall w, k \in \mathscr{W} : k < w, w \notin \mathscr{W}^{AP}. \tag{5c}$$

The problem is similar to (3), but adds constraints (5c) to ban waves not used in the *a priori* solution; function **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 a request arrives, and before each planned dispatch decision. The initial dispatch plan π for RP matches the optimal *a priori* plan. Define the expected number of requests realized after time $t \in \mathscr{T}$ at node $i \in I$ that are ready by wave

$w \in \mathscr{W}$ as

$$f_{i,w}(t) = \begin{cases} \mathbb{E}(N_i(t_w + p) - N_i(t)) & \text{if } t > t_w + p, \\ 0 & \text{otherwise;} \end{cases} \tag{6}$$

for a Poisson process with rate λ_i this becomes $f_{i,w}(t) = \lambda_i \max\{0, t - \max(t^{ct}, t_w + p)\}$. RP will update the dispatch plan in function **ArrivalUpdate**(RP, π, s, i, b) solving an optimal *a priori* plan for state s in (7), conditioned on a new request realization at node i,

$$\min_{\{\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{v},\mathbf{s}\} \in \mathscr{D}(w)} \beta_i\{z_i + \sum_{k=\max(b+1,h(0,i))}^{w} y_i^k\} + \sum_{k=1}^{w} \sum_{e \in E_k} \gamma d_e x_e^k + \sum_{j \in I} \beta_j \left\{ f_{j,0}(t) z_j + \sum_{k=h(0,j)}^{w} \left(f_{j,0}(t) - f_{j,k}(t) \right) y_j^k \right\} \tag{7a}$$

$$\text{s.t.} \sum_{k=h(j,0)}^{\min\{a_j,w\}} y_j^k = 1, \qquad \forall j \in I_{\mathbf{a}}, \tag{7b}$$

where the problem's domain equals (3), but it incorporates penalties for expected rejections of future request 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 **DispatchUpdate**(RP, π, s) solving

$$\min_{\{\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{v},\mathbf{s}\} \in \mathscr{D}(w)} \sum_{k=1}^{w} \sum_{e \in E_k} \gamma d_e x_e^k + \sum_{j \in I} \beta_i \left\{ f_{j,0}(t_w) z_j + \sum_{k=h(0,i)}^{w} \left(f_{j,0}(t_w) - f_{j,k}(t_w) \right) y_j^k \right\} \tag{8a}$$

$$\text{s.t.} \sum_{k=h(j,0)}^{\min\{a_j,w\}} y_j^k = 1, \qquad \forall j \in I_{\mathbf{a}}. \tag{8b}$$

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 **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 π feasible for the generic IP

$$GC(w,g) = \min_{\{\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{v},\mathbf{s}\}\in\mathscr{D}(w)} \sum_{k=1}^{w}\sum_{e\in E_k} \gamma d_e x_e^k + \sum_{i\in I}\left\{ g_{i,0}z_i + \sum_{k=h(0,i)}^{w} g_{i,k}y_i^k \right\}, \tag{9}$$

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: procedure RunHeuristic(Initial feasible plan $\pi^0$, maximum random destructions $k^{max}$)
2:     $\pi \leftarrow Copy(\pi^0)$, $\pi^* \leftarrow Copy(\pi^0)$
3:     do
4:         if ($\neg$IntraLS$(r,r^*)$ and $\neg$InterLS$(r,r^*)$ and $\neg$WavesLS$(r,r^*)$) then
5:             RandomDestruction$(r)$
6:     while (less than $k^{max}$ passes)
7:     return $r^*$
```

Algorithm 2 provides a high-level description of the heuristic; it requires an initial feasible plan π^0, and uses three neighborhood searches to improve upon the best available plan π^*: (1) intra-route local search (**IntraLS**), *i.e.*, single route node selection and re-sequencing; (2) inter-route local search (**InterLS**), *i.e.*, node exchanges between routes and re-sequencing; and (3) wave local search (**WavesLS**), *i.e.*, changes in the number of routes and dispatch times. Each local search returns $true$ if it has updated and accepted a new local solution π, and else, it returns $false$. 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 π; this is done k^{max} times before the heuristic outputs plan π^*. 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 *RunHeuristic* a second time and recursively over each candidate plan π' that does not improve the local solution π after one local search move $\pi \to \pi'$, but has a small enough cost to be a good candidate to start a subsequent meta-heuristic search.

Based on this heristic, we propose a Heuristic Acceptance Rollout Policy (HARP) in which the initial dispatch plan (**IniPlan**) and the update before dispatch decisions (**DispatchUpdate**) match RP's functions, but it heuristically solves (7) instead of an IP solver to update the plan upon request arrivals in **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 *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 $d^{max} = \frac{2T}{7}$, 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 $g \in \{0,\dots,4\}$, 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 ℓ_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 request 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 $p_{start} = \{0\%, 15\%, 30\%\}$ 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_{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 λ_i defining the Poisson process at node $i \in I$ is randomly generated in clusters of 10 nodes so that $10\lambda^* = (T - t^{ct}) \sum_{i=10(k-1)}^{10k} \lambda_i$ 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 $(g, p_{start}, s, \lambda^*) : g \in \{0,1,2,3,4\}, p_{start} \in \{0\%, 15\%, 30\%\}, s \in \{0,1,2\}, \lambda^* \in \{0.5, 1, 2\}$.

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

metric	**definition**
$cost/request$	the total cost divided by the total number of requests (initial & realized)
$cost/tt$	the total cost divided by the vehicle's total travel time; this metric is one or greater, since total cost is the sum of total travel time and penalties paid.
fill rate (fr)	the percentage of requests accepted by the vehicle over all realized requests
travel time per request served ($ttprs$)	the policy's vehicle travel time over the total number of requests served
gap^P	the percentage increase of the policy's cost over $P \in \{LB, FLEX, RP\}$, respectively
$nDispatches$	number of vehicle routes dispatched
$nWaves$	average number of waves used by each vehicle route dispatched over the realization
$iWait$	number of waves spent waiting at the depot before the initial dispatch
$afterCT$	percentage of orders served that are dispatched after the cut-off time
$nodes/dispatch$	average number of node visits per route dispatched
$time_{off}$	average off-line initial plan construction time, *i.e.*, a call of **IniPlan**
$time_{dis}$	average plan update time before dispatch decisions, *i.e.*, a call of **DispatchUpdate**
$time_{acc}$	average plan update time before acceptance decisions, *i.e.*, a call of **ArrivalUpdate**

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 different behavior. MP keeps travel time per request served low taking advantage of re-optimization capabilities. However, its myopic behavior does not generate enough vehicle returns for recourse possibilities, producing fewer dispatches, longer dispatch duration, and more average nodes per route dispatched than LB. Unlike MP, the *a priori* policy (AP) becomes inefficient in travel time per request served 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

metric \policy	**LB**	**FLEX**	**MP**	**AP**	**MPF**	**RP**	**HARP**
$cost/request$	11.5	13.3	15.4	17.0	15.2	13.9	14.2
$cost/tt$	1.53	1.97	2.52	2.64	2.41	2.07	2.15
fr	93.4%	88.6%	85.5%	80.1%	85.6%	87.8%	86.9%
$ttprs$	9.1	8.8	8.9	9.4	9.0	8.9	8.9
gap^{LB}	N/A	15.9%	35.0%	48.0%	33.0%	21.0%	23.8%
gap^{FLEX}	N/A	N/A	16.2%	27.1%	14.5%	4.4%	6.8%
gap^{RP}	N/A	-4.2%	11.2%	21.8%	9.7%	N/A	2.3%
$time_{off}$ (sec.)	121.9	529.2	0.00	529.2	529.2	529.2	529.2
$time_{disp}$ (sec.)	0.00	81.8	0.00	0.00	0.00	68.6	80.1
$time_{acc}$ (sec.)	0.00	0.00	2.4	0.00	1.2	18.1	1.1
$nDispatches$	2.69	2.51	1.94	2.48	2.21	2.52	2.52
$iWait$	2.87	3.21	3.43	3.09	3.35	3.20	3.21
$nWaves$	1.61	1.58	1.86	1.64	1.72	1.59	1.58
$nodes/dispatch$	7.2	7.4	9.2	6.7	8.3	7.4	7.3
$afterCT$	68.4%	79.0%	73.3%	78.3%	76.3%	78.7%	79.1%

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 request acceptance, which is crucial for companies interested in providing the best possible customer service. RP has the lowest cost per time traveled over all tested feasible policies, meaning it has the smallest ratio of penalty over vehicle operating cost. The average $cost/tt$ metric it is only smaller for the infeasible FLEX policy and the PIR, which are closer to 1. 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

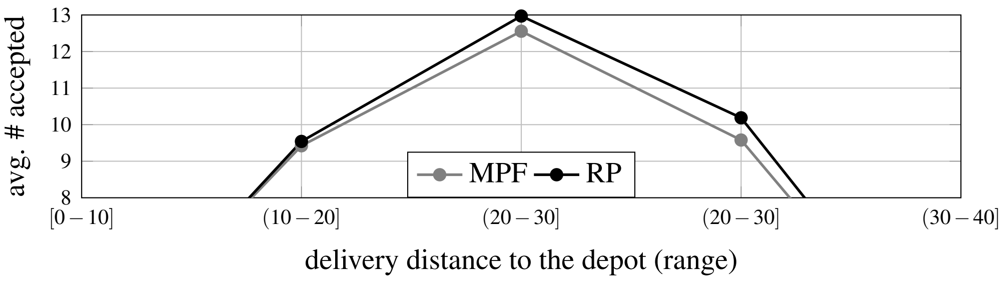


Figure 6: Average number of orders accepted over each orders' distance to the depot range.

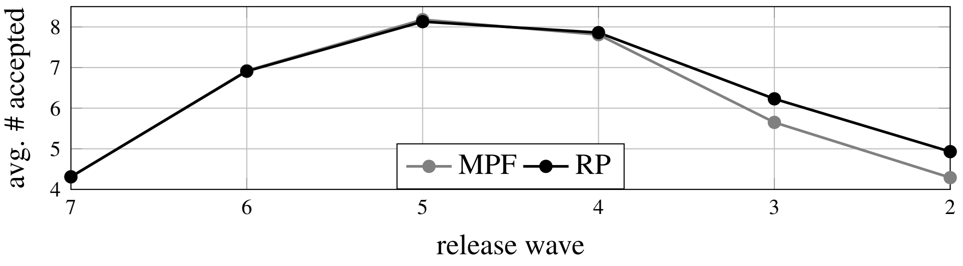


Figure 7: Average number of orders accepted over the orders' earliest dispatch wave.

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

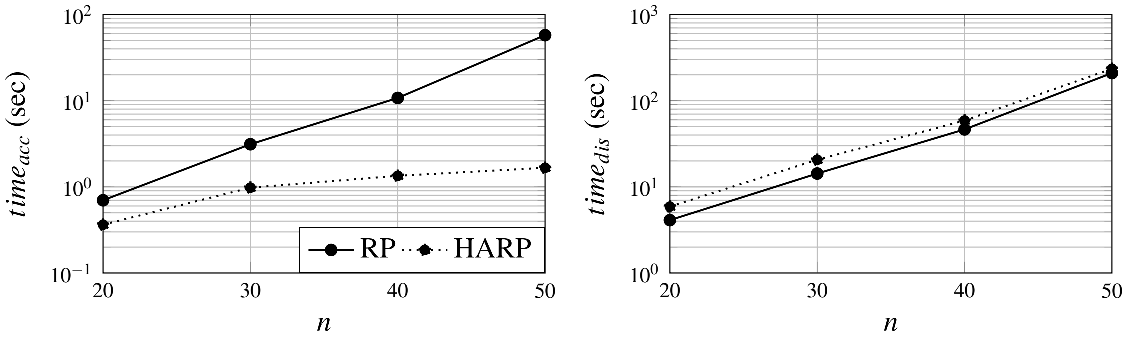


Figure 8: Average time per request acceptance decision and average time per dispatch decision versus instance size in number of nodes (n)

All policies computing an initial *a priori* solution share offline computation time ($time_{off}$), but differ in online computation per dispatch ($time_{disp}$) and per request acceptance decision ($time_{acc}$); 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 $time_{acc}$ and $time_{disp}$ 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 $time_{acc}$, 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.

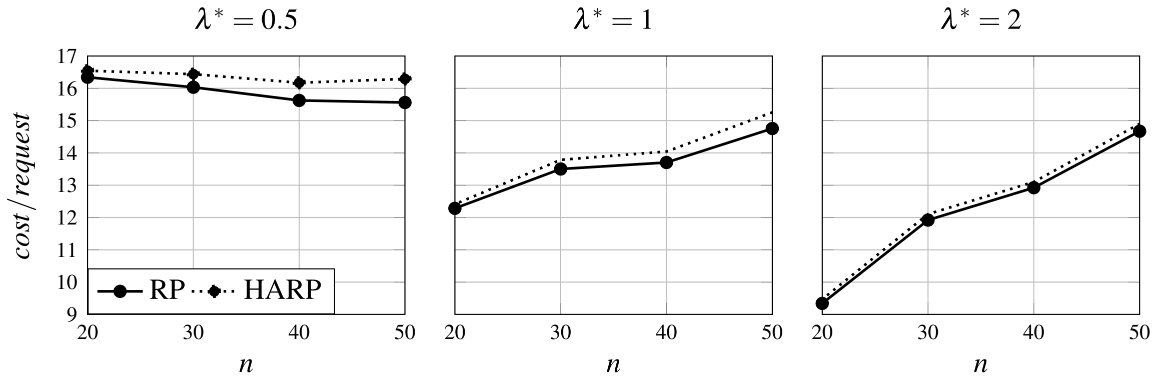


Figure 9: Average *cost*/*request* versus number of nodes (n) and online arrival intensity (λ^*)

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 ($\lambda^* = 0.5$), possibly due to service consolidation. Conversely, the cost per request 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.

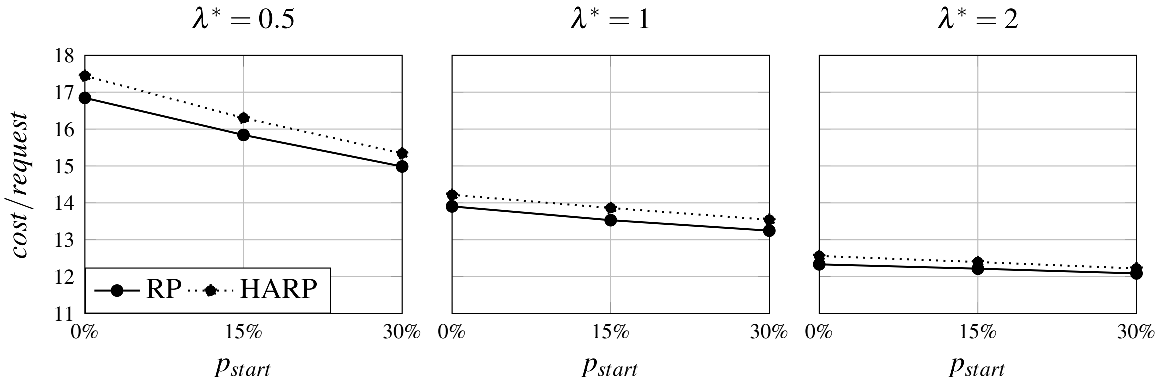


Figure 10: Average $cost/request$ versus offline request probability (p_{start}) and online arrival intensity (λ^*)

Figure 10 presents the average cost per request of RP and HARP as a function of the probability of having orders waiting for service before execution starts (p_{start}) and the average arrival intensity (λ^*). 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_{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 request tends to stabilize for instances with $\lambda^* = 2$, where off-line orders lose relative importance.

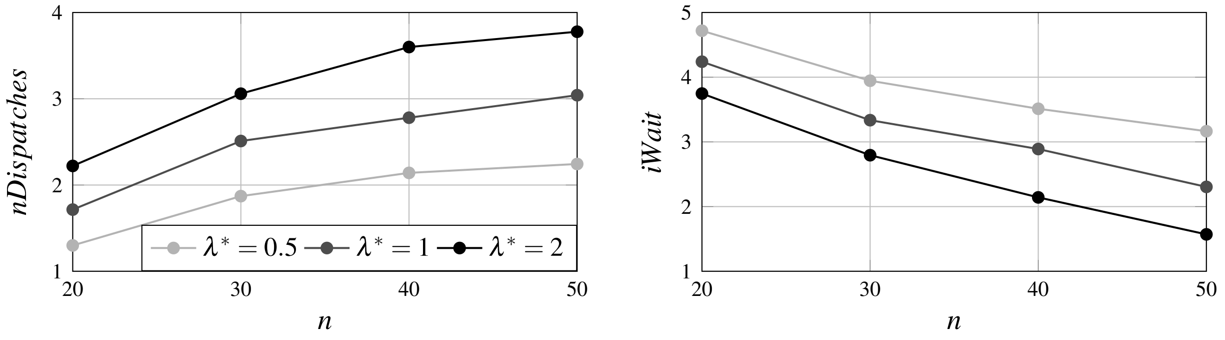


Figure 11: Average number of routes dispatched and waves waited before dispatch ($iWait$) for RP policy versus number of nodes (n) and online arrival intensity (λ^*)

Figure 11 presents the average number of routes dispatched and initial wait at the depot ($iWait$) for RP as a function of n and λ^*. For a relatively busier instance (bigger n and λ), 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 dispatches and wait more at the depot than instances with scattered and bigger networks

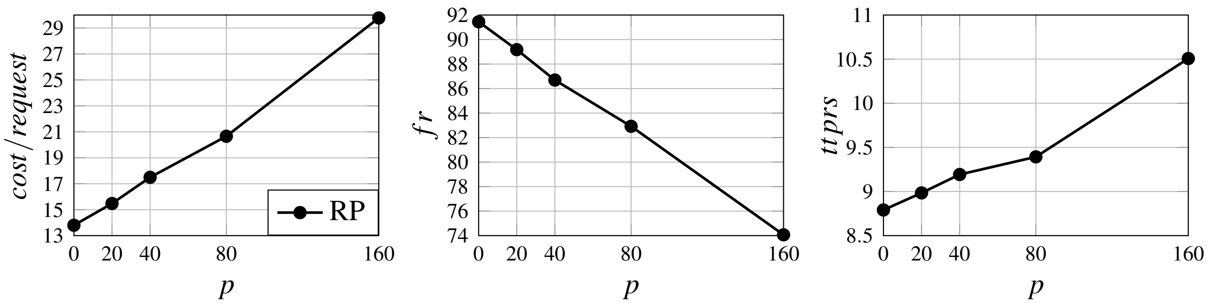


Figure 12: Average $cost/request$, fr, and $ttprs$ versus request processing time (p)

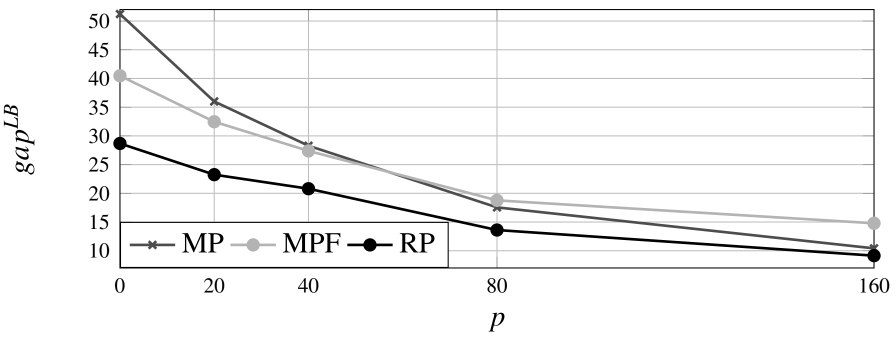


Figure 13: Average gap^{LB} versus request processing time (p)

($n = 40, \lambda^* = 1$).

7.3 Analysis of performance sensitivity over processing time

Now we study the sensitivity of RP to the request 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 request 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 request 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 request 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 instances with an equal number of request arrivals per day, we keep all originally simulated request 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})

t^{ct}	$cost/request$	$gap^{LB}(\%)$	$fr(\%)$	$ttprs$	$afterCT(\%)$
126	27.4	21.8	76.2	10.1	21.0
252	17.1	32.4	86.8	8.9	65.0
378	9.9	16.0	95.7	7.5	94.8

7.5 Analysis of performance sensitivity under varying penalty cost levels

Finally, we study the performance of our best dynamic policies as a function of the relative magnitude of the penalty paid per rejected request. To do so we solve all previous instances with intermediate problem sizes ($n = 30$ and $\lambda^* = 1$) using five different settings of the penalty cost function $\beta_i(\alpha) = \alpha \cdot d_{0,i} + 1$, for each $\alpha \in \{1,2,4,8,100\}$. While the setting $\alpha = 1$ assigns a relatively small request rejection cost similar to the customer's distance to the depot, the value $\alpha = 100$ hierarchically focuses on accepting requests first and then minimizing routing costs; the setting $\alpha = 2$ is the base case setting request rejection costs similar to direct dispatch costs, and $\alpha = 4$ and 8 are intermediate cases.

A change of penalty magnitudes distorts the objective function and, thus, one cannot compare these solutions in terms of $cost/request$. A fairer comparison is presented in Table 4 for the RP policy in terms of what we consider the essential objectives of any delivery operation: (1) effectiveness (measured in customer fill rate) and (2) efficiency (measured in vehicle travel time per request served). As discussed in [25], there exists a trade-off between customer fill rate and vehicle routing costs per request serviced, meaning that an increase in customer acceptance rates requires a sacrifice in vehicle routing efficiency. Also, this sacrifice is marginally increasing as customer fill rate increases. When the focus is accepting requests, our *RP* policy adapts and becomes closer to a reactive direct dispatch operation increasing the average number of dispatches per vehicle, reducing the number of locations visited per dispatch and dispatching earlier in the day. Conversely, when efficiency is relatively more important there are fewer vehicles dispatches and they occur later in the day and visit more customer locations.

Table 4: Average performance indicators of RP versus varying penalty cost levels (α)

metric\\α	1	2	4	8	100
fr	87.1%	89.2%	90.0%	90.7%	91.5%
$ttprs$	8.4	9.0	9.4	10.1	11.6
$cost/tt$	1.48	1.72	2.25	3.13	22.05
$nDispatches$	2.10	2.51	2.82	3.14	3.98
$iWait$	3.61	3.34	3.10	2.78	1.86
$nWaves$	1.66	1.50	1.41	1.36	1.27
$nodes/dispatch$	7.8	6.7	6.1	5.6	4.5

We provide Pareto charts for each policy's average fr and $ttprs$ in Figure 14. The increasing slopes of these curves empirically verify an increasing marginal sacrifice of $ttprs$ per additional percentage increase in fr. We also observe how the *RP* policy manages to cut the gap between the *a priori* policy (AP) and the perfect information bound by more than half over any type of objective. Finally, we empirically observe the sacrifice made by the system when imposing immediate request acceptance, *i.e.*, the difference between the curves of RP and FLEX. This difference becomes smaller as fill rate becomes relatively more important than routing costs.

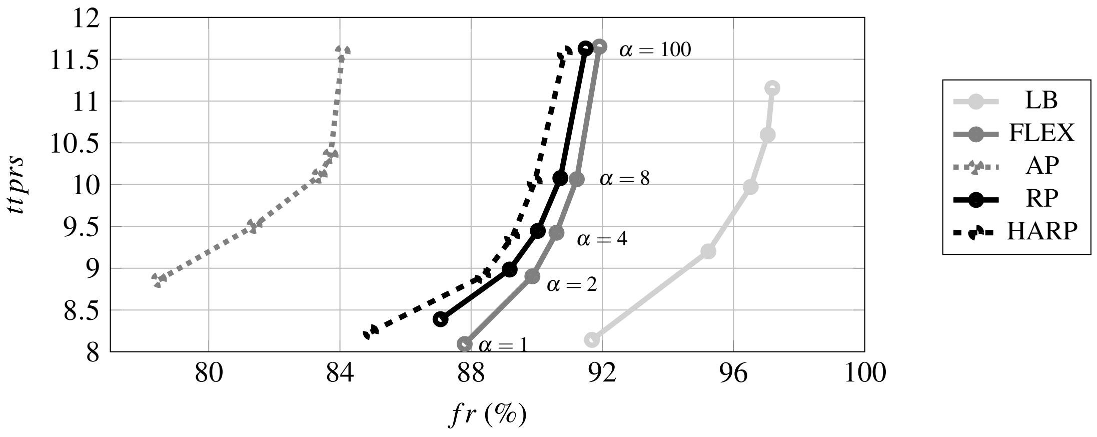


Figure 14: Pareto charts showing each policy's average fr and $duration/order$ for different settings α

8 Conclusions

We formulate the DDWP-IA for SDD operations, which integrates immediate request acceptance and processing with distribution to the customer. 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 ir to benchmark policies and relaxed policies over a set of computational instances under different settings of geography, problem size, online request arrival intensity, percentage of accepted orders known before the operation starts, and SDD objectives varying the relative importance of customer fill rate over routing costs. RP outperforms any of our feasible 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 request 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 request 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 request processing times may be directly transferred to a reduction in cost per request in the distribution operation 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 request 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, 49] for other dynamic routing problems.

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

Acknowledgment

We thank the referees and editors for their valuable comments.

References

[1] N. Agatz, A. M. Campbell, M. Fleischmann, and M. W. P. Savelsbergh, *Time slot management in attended home delivery*, Transportation Science **45** (2011), no. 3, 435–449.

[2] M. Albareda-Sambola, E. Fernández, and G. Laporte, *The dynamic multiperiod vehicle routing problem with probabilistic information*, Computers & Operations Research **48** (2014), no. 0, 31–39.

[3] C. Archetti, D. Feillet, and M.G. Speranza, *Complexity of routing problems with release dates*, European Journal of Operational Research **247** (2015), no. 3, 797 – 803.

[4] N. Azi, M. Gendreau, and J.-Y. Potvin, *An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles*, European Journal of Operational Research **202** (2010), no. 3, 756 – 763.

[5] ______, *A dynamic vehicle routing problem with multiple delivery routes*, Annals of Operations Research **199** (2012), no. 1, 103–112.

[6] ______, *An adaptive large neighborhood search for a vehicle routing problem with multiple routes*, Computers & Operations Research **41** (2014), 167 – 173.

[7] E. Balas, *The prize collecting traveling salesman problem*, Networks **19** (1989), no. 6, 621–636.

[8] R. Bent and P. van Hentenryck, *Scenario-based planning for partially dynamic vehicle routing with stochastic customers*, Operations Research **52** (2004), no. 6, 977–987.

[9] D. Bertsekas, *Dynamic programming and optimal control*, vol. 1, Athena Scientific Belmont, MA, 1995.

[10] D. Brown, J. Smith, and P. Sun, *Information relaxations and duality in stochastic dynamic programs*, Operations research **58** (2010), 785–801.

[11] A. M. Campbell and M. W. P. Savelsbergh, *Decision support for consumer direct grocery initiatives*, Transportation Science **39** (2005), no. 3, 313–327.

[12] A.M. Campbell and B. W. Thomas, *Probabilistic traveling salesman problem with deadlines*, Transportation Science **42** (2008), no. 1, 1–21.

[13] D. Cattaruzza, N. Absi, and D Feillet, *The multi-trip vehicle routing problem with time windows and release dates*, Transportation Science **50** (2016), no. 2, 676–693.

[14] D. Cattaruzza, N. Absi, D. Feillet, and J. González-Feliu, *Vehicle routing problems for city logistics*, EURO Journal on Transportation and Logistics (2015), 1–29.

[15] J. F. Cordeau, G. Laporte, M. W. P. Savelsbergh, and D. Vigo, *Vehicle routing*, Transportation, handbooks in operations research and management science **14** (2006), 367–428.

[16] A. Erera, M. W. P. Savelsbergh, and E. Uyar, *Fixed routes with backup vehicles for stochastic vehicle routing problems with time constraints*, Networks **54** (2009), no. 4, 270–283.

[17] A.J.R.M. Gademann, J. P. Van Den Berg, and H. H. Van Der Hoff, *An order batching algorithm for wave picking in a parallel-aisle warehouse*, IIE Transactions **33** (2001), no. 5, 385–398.

[18] M. Gendreau, G. Laporte, and R. Séguin, *Stochastic vehicle routing*, European Journal of Operational Research **88** (1996), no. 1, 3–12.

[19] J. C. Goodson, J. W. Ohlmann, and B. W. Thomas, *Rollout policies for dynamic solutions to the multivehicle routing problem with stochastic demand and duration limits*, Operations Research **61** (2013), no. 1, 138–154.

[20] J. C. Goodson, B. W. Thomas, and J. W. Ohlmann, *Restocking-based rollout policies for the vehicle routing problem with stochastic demand and duration limits*, Transportation Science **50** (2016), no. 2, 591–607.

[21] ______, *A rollout algorithm framework for heuristic solutions to finite-horizon stochastic dynamic programs*, European Journal of Operational Research **258** (2017), no. 1, 216 – 229.

[22] B. Hochfelder, *What retailers can do to make the last mile more efficient*, 5 2017, `http://www.supplychaindive.com/news/last-mile-spotlight-retail-costs-fulfillment/443094/` (accesed on May 19, 2017).

[23] D. Ingold and S. Soper, *Amazon doesn't consider the race of its customers. should it?*, 4 2016, `https://www.bloomberg.com/graphics/2016-amazon-same-day/` (accesed on May 19, 2017).

[24] P. Jaillet, *A priori solution of a traveling salesman problem in which a random subset of the customers are visited*, Operations Research **36** (1988), no. 6, 929–936.

[25] M. Klapp, A. Erera, and A Toriello, *The dynamic dispatch waves problem for same-day delivery*, European Journal of Operational Research. **271** (2018), no. 2, 519–534.

[26] ———, *The one-dimensional dynamic dispatch waves problem*, Transportation Science **52** (2018), no. 2, 402–415.

[27] G. Laporte, F. Louveaux, and H. Mercure, *A priori optimization of the probabilistic traveling salesman problem*, Operations Research **42** (1994), no. 3, 543–549.

[28] A. Larsen, O.B.G. Madsen, and M.M. Solomon, *Recent developments in dynamic vehicle routing systems*, The Vehicle Routing Problem: Latest Advances and New Challenges, Springer, 2008, pp. 199–218.

[29] U.S. Department of Commerce, *U.s. census bureau news: quaterly retail e-commerce sales - first quarter 2018*, 5 2018, pp. 1–3.

[30] V. Pillac, M. Gendreau, C. Guéret, and A. Medaglia, *A review of dynamic vehicle routing problems*, European Journal of Operational Research **225** (2013), no. 1, 1–11.

[31] W. Powell, *Approximate dynamic programming: Solving the curses of dimensionality*, John Wiley & Sons, 2007.

[32] M. Puterman, *Markov decision processes: discrete stochastic dynamic programming*, John Wiley & Sons, 2009.

[33] M. W. P. Savelsbergh and Sol M., *The general pickup and delivery problem*, Transportation Science **29** (1995), no. 1, 17–29.

[34] M. W. P. Savelsbergh and T. Van Woensel, *50th anniversary invited article - city logistics: Challenges and opportunities*, Transportation Science **50** (2016), no. 2, 579–590.

[35] N. Secomandi and F. Margot, *Reoptimization approaches for the vehicle-routing problem with stochastic demands*, Operations Research **57** (2009), no. 1, 214–230.

[36] B. Thomas, *Dynamic vehicle routing*, Wiley Encyclopedia of Operations Research and Management Science, John Wiley and Sons, Inc., 2010, pp. 1–11.

[37] P. Toth and D. Vigo, *The granular tabu search and its application to the vehicle-routing problem*, Informs Journal on computing **15** (2003), no. 4, 333–346.

[38] ———, *Vehicle routing: Problems, methods, and applications*, vol. 18, SIAM, 2014.

[39] M. W. Ulmer, *Approximate dynamic programming for dynamic vehicle routing*, vol. 61, Springer, 2017.

[40] M. W. Ulmer, *Dynamic pricing for same-day delivery routing*, Technical Report, Technical University Braunschweig, Braunschweig (2017), 1–34.

[41] M. W. Ulmer, J. C. Goodson, D. C. Mattfeld, and B. W. Thomas, *Route-based markov decision processes for dynamic vehicle routing problems*, Tech. report, Working Paper, Technical University Braunschweig, Braunschweig, 2017.

[42] M. W. Ulmer, D. C. Mattfeld, M. Hennig, and J. C. Goodson, *A rollout algorithm for vehicle routing with stochastic customer requests*, Logistics Management, Springer, 2016, pp. 217–227.

[43] M. W. Ulmer, D. C. Mattfeld, and F. Köster, *Budgeting time for dynamic vehicle routing with stochastic customer requests*, Transportation Science **52** (2018), no. 1, 20–37.

[44] M. W. Ulmer, D. C. Mattfeld, and N. Soeffker, *Dynamic multi-period vehicle routing: approximate value iteration based on dynamic lookup tables*, Technical Report, Technische Universität Braunschweig (2016), 1–17.

[45] M. W. Ulmer, N. Soeffker, and D. C. Mattfeld, *Value function approximation for dynamic multi-period vehicle routing*, European Journal of Operational Research **269** (2018), no. 3, 883 – 899.

[46] M. W. Ulmer, B. W. Thomas, and D. C. Mattfeld, *Preemptive depot returns for a dynamic same-day delivery problem*, EURO Journal on Transportation and Logistics (2018), 1–35.

[47] W. J. A. van Heeswijk, M. R. K. Mes, and J. M. J. Schutten, *The delivery dispatching problem with time windows for urban consolidation centers*, published online on Transportation Science (2017), 1–19.

[48] S. Voccia, A.M. Campbell, and B. W. Thomas, *The probabilistic traveling salesman problem with time windows*, EURO Journal on Transportation and Logistics (2012), 1–19.

[49] S. Voccia, A.M. Campbell, and B.W. Thomas, *The same-day delivery problem for online purchases*, published online on Transportation Science (2015), 1–18.

[50] M. Wen, J.-F. Cordeau, G. Laporte, and J. Larsen, *The dynamic multi-period vehicle routing problem*, Computers & Operations Research **37** (2010), no. 9, 1615–1623.

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 π structure and searches to improve the best plan available so far π^*.

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

$$PCTSP(d^{max}, Q, \rho) := \min_{S \subseteq Q:\, t^*(S) \leq d^{max}} \left\{ c^*(S) - \sum_{i \in S} \rho_i \right\} \tag{10}$$

solved over a subset $Q \subseteq I$ of nodes, prizes $\rho_i, i \in Q$, and a maximum route duration d^{max}. IntraLS is a best move procedure, where a move is described by re-optimizing one route r_w^π dispatched at wave w from the local solution π. Let π' be a copy of the local solution π after removing route r_w^π from it and leaving all remaining routes unaltered. The procedure solves a PC-TSP over the set of nodes in π' left unattended $\bar{I}(\pi') = \{i \in I : i \notin r_k^{\pi'}, \forall k \in \mathscr{W}^{\pi'}\}$, a maximum route duration equal to the duration of the waves left available after removing route r_w^π, 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 π after each best improvement loop if the best move candidate $\hat{\pi}$ has a lower cost; it also updates the overall best solution π^*. The procedure returns a boolean variable with a $true$ value if the local plan π was improved and returns $false$ if not. Any local solution π processed by IntraLS contains only routes $r_k^\pi, k \in \mathscr{W}^\pi$ 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: procedure IntraLS(plan $\pi$, best plan $\pi^*$)
2:     $\mu \leftarrow false$
3:     loop
4:         $\hat{\pi} \leftarrow \pi$          //initialize best candidate
5:         for $w \in \mathscr{W}^\pi$ do
6:             Let $\pi'$ be a copy of $\pi$ without route $r_w^\pi$
7:             Let $d^{max} \leftarrow t_w - t_{q(w, r_w^\pi)}$
8:             Solve PCTSP($d^{max}, \bar{I}(r'), \{g_{i,0} - g_{i,w}\}$) and add the optimal route found to $\pi'$ and dispatch it
   at wave $w$
9:             if ($c_{\pi'} < c_{\hat{\pi}}$) then $\hat{\pi} \leftarrow \pi'$          //update best candidate
10:        if ($c_{\hat{\pi}} < c_\pi$) then
11:            $\pi \leftarrow \hat{\pi}$, $\mu \leftarrow true$          //update local solution
12:            if ($c_{\hat{\pi}} < c_{\pi^*}$) then $\pi^* \leftarrow \hat{\pi}$          //update best solution
13:        else break loop
14:    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

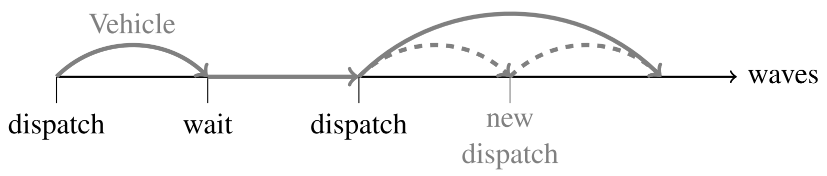


Figure 15: Example of a cut operation where a new dispatch plan is created (dashed flow) from an existing one (continuous flow) by adding an extra return to the depot.

the prize-collecting component; a move changes penalty savings (due to the different dispatch time). Second, we check the duration 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 π, the best solution π^* and returns *true* if the local solution π 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 $\mathscr{W}^{\pi}$ of a plan π 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-reassign the routes dispatched in π to the best possible dispatch waves, without altering the customer visit sequences or route duration. The last six search over new candidate solutions by changing the dispatch structure of π and solving multiple PC-TSPs.

Algorithm 4 Waves Local Search (WavesLS)

```
1: procedure WAVESLS(local plan $\pi$, best plan $\pi^*$)
2:     loop
3:         if ($\neg$REORDER($\pi,\pi^*$) and $\neg$CUT($\pi,\pi^*$) and $\neg$MERGE($\pi,\pi^*$) and $\neg$INSERT($\pi,\pi^*$) and $\neg$DELETE($\pi,\pi^*$) and $\neg$ENLARGE($\pi,\pi^*$)and $\neg$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^{π}_{w} with duration $w - q(w, r^{\pi}_{w}) \geq 2$ waves into two dispatches with shorter wave duration; this operator adds an extra return trip to the depot, as depicted in Figure 15. 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 16. The Insert operator, described in Algorithm 7, inserts a new route with duration one wave between two dispatched routes in a plan π shifting all previous dispatches a wave earlier in time; see Figure 17. 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 18. 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 19. 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

Algorithm 5 Cut operation

```
1:  procedure CUT(local plan $\pi$, best plan so far $\pi^*$)
2:      for $w \in \mathscr{W}^\pi$ do
3:          $d^{max} \leftarrow t_w - t_{q(w,r_w^\pi)}$
4:          for $v : (w-1) \rightarrow (w - q(w,r_w^\pi)+1)$ do
5:              Let $\pi'$ a copy of $\pi$ without route $r_w^\pi$
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:              Solve PCTSP($d^{max} - (t_w - t_v), \bar{I}(\pi'), \{g_{i,0} - g_{i,v}\}$) and add optimal route to $\pi'$ at wave $v$
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:     return $false$
```

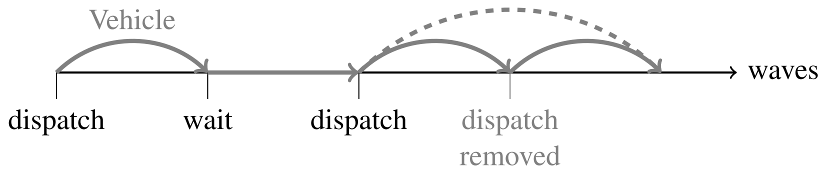


Figure 16: Example of a merge operation where a new dispatch plan is created (dashed flow) from an existing one (continuous flow) by removing one return to the depot and merging two dispatches.

Algorithm 6 Merge operation

```
1:  procedure MERGE(local plan $\pi$, best plan $\pi^*$)
2:      for $w \in \mathscr{W}^\pi$ such that $q(w,r_w^\pi) > 0$ do
3:          Let $w' \leftarrow q(w,r_w^\pi)$
4:          Let $\pi'$ be a copy of $\pi$ without routes $r_w^\pi$ and $r_{w'}^\pi$
5:          Let $d^{max} \leftarrow t_w - t_{q(w',r_{w'}^\pi)}$
6:          Solve PCTSP($d^{max}, \bar{I}(\pi'), \{g_{i,0} - g_{i,w}\}$) 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:     return $false$
```

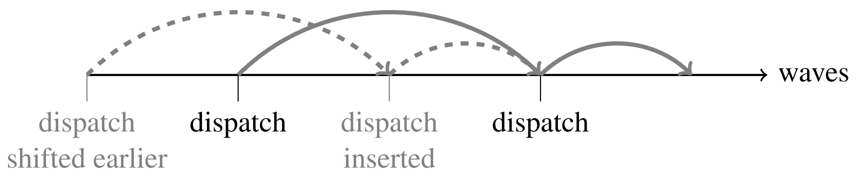


Figure 17: Example of an Insert operation where an new dispatch is inserted launching previous routes a wave earlier (dashed flow).

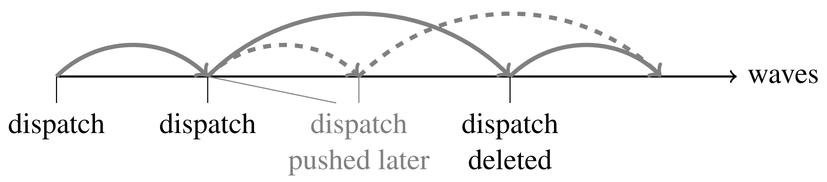


Figure 18: Example of a Delete operation where last dispatch is deleted pushing preceding ones later (dashed flow).

executing all precedent dispatches one wave later, see Figure 20.

Algorithm 7 Insert operation

```
1: procedure INSERT(local plan $\pi$, best plan $\pi^*$)
2:     if $(\max\{k \in \mathscr{W}^\pi\} = W)$ then return $false$
3:     for $w \in \mathscr{W}^\pi \cup \{0\}$ do
4:         Let $\pi'$ a copy of $\pi$ with routes $r_k^\pi, k \in \mathscr{W}^\pi : k > w$ 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 $w \in \mathscr{W}^\pi$ do
3:         Let $\pi'$ a copy of $\pi$ with $r_w^\pi$ deleted and all routes $r_k^\pi, k \in \mathscr{W}^\pi : k > w$ dispatched $w - q(w, r_w^\pi)$ 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 π to move the search to distant solutions and avoid local optimality. This

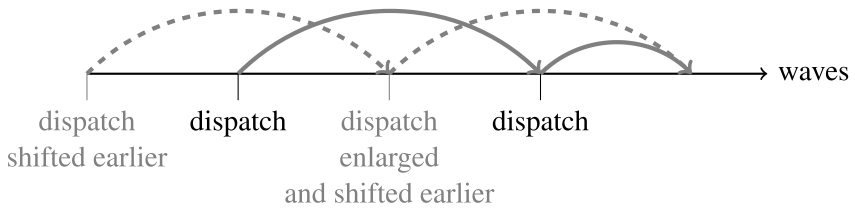


Figure 19: Example of an Enlarge operation where the last dispatch is enlarged and launched a wave earlier pushing the previous dispatch earlier too (dashed flow).

Algorithm 9 Enlarge operation

```
1: procedure ENLARGE(local plan $\pi$, best plan $\pi^*$)
2:     if $(\max\{k \in \mathscr{W}^\pi\} = W)$ then return $false$
3:     for $w \in \mathscr{W}^\pi$ do
4:         Let $\pi'$ a copy of $\pi$ without route $r^\pi_w$ and all routes $r^\pi_k, k \in \mathscr{W}^\pi : k > w$ dispatched one wave earlier.
5:         Solve PCTSP($t_{w+1} - t_{q(w,r^\pi_w)}, \bar{I}(\pi'), \{g_{i,0} - g_{i,w+1}\}$) 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 $false$
```

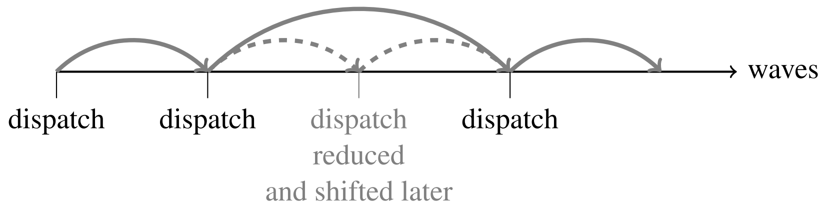


Figure 20: Example of a Reduce operation where one dispatch is reduced and launched one wave later pushing the previous dispatch later too (dashed flow).

Algorithm 10 Reduce operation

```
1: procedure REDUCE(local plan $\pi$, best plan $\pi^*$)
2:     for $w \in \mathscr{W}^\pi$ such that $w - q(w, r^\pi_w) > 1$ do
3:         Let $\pi'$ a copy of $\pi$ without route $r^\pi_w$ and all routes $r^\pi_k, k \in \mathscr{W}^\pi : k > w$ dispatched one wave later.
4:         Solve PCTSP($t_{w-1} - t_{q(w,r^\pi_w)}, \bar{I}(\pi'), \{g_{i,0} - g_{i,w-1}\}$) 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 π

```
1:  procedure RandomDestruction(local solution $\pi$)
2:      Generate a random number $x \in \{1, \dots, \lfloor 0.5|\mathscr{W}^{\pi}| \rfloor\}$
3:      Randomly delete $x$ routes from plan $\pi$.
4:      if ($\pi$ is empty) then return
5:      for $(w \in \mathscr{W}^{\pi})$ do
6:          Set $y$ equal to the number of nodes visited in $r_w^{\pi}$.
7:          if $(y > 1)$ then
8:              Generate a random number $z \in \{\lceil 0.25y \rceil, \dots, \lfloor 0.75y \rfloor\}$.
9:              Randomly delete and skip $z$ visits from $r_w^{\pi}$
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 $\rho_i, i \in Q$, and maximum duration d^{max}. This solution implements simulated annealing running over an elementary route $r = \{0, i_1, i_2, .., 0\}$ with objective value v_p with a subset $Q^r_{out} \subseteq Q$ of unattended nodes.

The parameters T_0 and δ 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 $\mathscr{N}(r)$ 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 $i \in Q^r_{out}$ and a visited node j in route r ($\mathscr{O}(|Q|^2)$ moves),
2. an insertion of an unvisited node $i \in Q^r_{out}$ after a visited node j in route r ($\mathscr{O}(|Q|^2)$ moves),
3. a removal of a visited node j from route r ($\mathscr{O}(|Q|)$ moves),
4. a removal of a visited node k from route r and the insertion of an unvisited node $i \in Q^r_{out}$ after a visited node j in route r ($\mathscr{O}(|Q|^3)$ moves),
5. 2-opt, *i.e.*, 2-edge exchanges within route r ($\mathscr{O}(|Q|^2)$ 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 $r_i, .., r_{i+k}$ after r_j ($\mathscr{O}(|Q|^3)$ moves),
7. a internal swap in route r between two visited nodes ($\mathscr{O}(|Q|^2)$ moves),
8. a swap between one visited node i in route r an two nodes $j,k \in Q_{out}$ inserted in series ($\mathscr{O}(|Q|^3)$ moves),
9. a swap between two consecutive visited nodes i, j in route r an one node $k \in Q_{out}$ ($\mathscr{O}(|Q|^2)$ moves),

Algorithm 12 Metaheuristic for the PC-TSP

```
1: procedure HPCTSP($d^{max}, Q, \rho, k^{max}$),
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 $Q$ int the last position of $r$; stop
   before violating the route duration constraint,
6:         Update the set of non-visited nodes $Q_{out}^{r'} \leftarrow Q$,
7:         Initialize temperature $T \leftarrow T_0$ and iteration counter $k \leftarrow 0$.
8:         while ($k < k^{max}$) do
9:             $update \leftarrow false$
10:            for (neighbor $r' \in \mathcal{N}(r)$) do
11:                Generate a random number $p$ for a $Uniform(0,1)$ distribution,
12:                if ($e^{(v_{r'} - v_r)/T} > p$) then
13:                    $r \leftarrow r'$, update $Q_{out}^{r}$, $update \leftarrow true$, and break for.
14:            if ($\neg update$) then Randomly skip and remove 30% of nodes from $r$. Update $Q_{out}$.
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 $r^*$
```

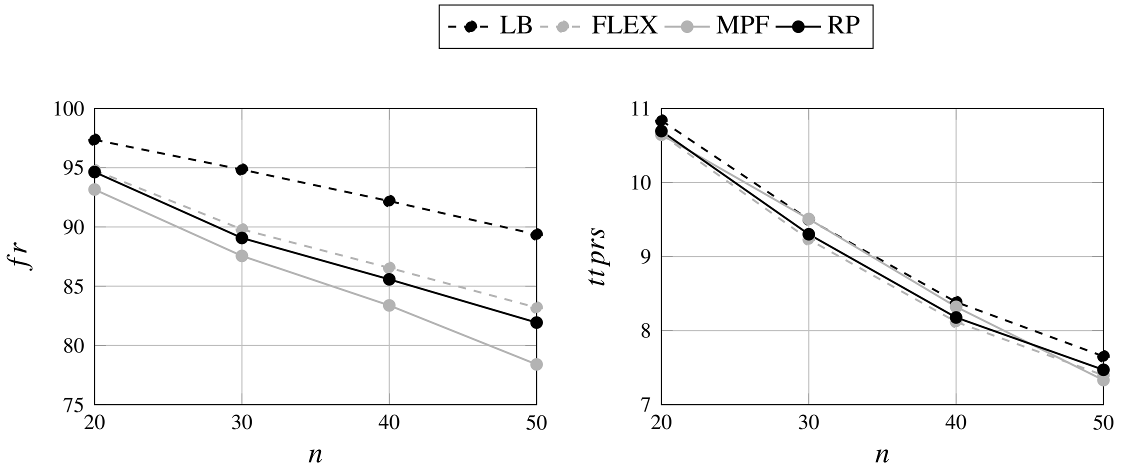


Figure 21: Average fill rate (fr) and travel time per request served for RP policy accepted versus number of nodes (n)

10. a swap between two consecutive visited nodes i, j in route r and two nodes $k, q \in Q_{out}$ inserted in series ($\mathscr{O}(|Q|^3)$ moves).

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

A.4 Average heuristic performace over instance size

In Figure 21 we show the performance of RP and MPF in terms of request fill rate (fr) and travel time per request served ($ttprs$) 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 ($ttprs$), 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 request arrivals for $t^{ct} = 252$ and distribute them proportionally between T and the new cut-off time. Formally, an arrival at $x_0 \in [t_0^{ct}, T]$ is relocated to time $x_1 = t_1^{ct} + \frac{T - t_1^{ct}}{T - t_0^{ct}}(x_0 - t_0^{ct}) \in [t_1^{ct}, T]$ when varying the cutoff time from t_0^{ct} to t_1^{ct}. Figure 22 presents an example with three requests relocated when t^{ct} changes from 252 to 126 and to 378.

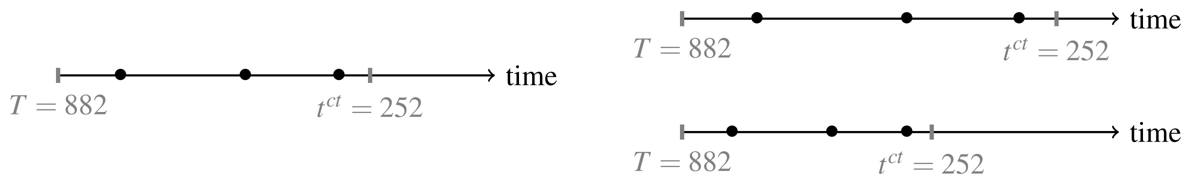


Figure 22: Example of request arrivals re-scaled from $t^{ct} = 252$ to $t^{ct} = 126$ and to $t^{ct} = 378$