Splitting shopping and delivery tasks in an on-demand personal shopper service

Alp Arslan∗, Niels Agatz† and Mathias Klapp‡

July 30, 2019

Abstract

We introduce a personal shopper service that offers the same-day delivery of products available at brick and mortar stores. Each customer request is placed online and could involve items from multiple stores. The operator of this service has to dynamically accept requests, coordinate a fleet of shoppers, schedule shopping operations at stores and execute deliveries of accepted requests to customers on time. Our work focuses on studying the benefits of splitting requests into smaller tasks served by different shoppers in parallel and consolidating tasks with a common shopping location into one store visit. We present and solve an online optimization model and empirically show that the percentage of requests served increases when request splitting is allowed. This benefit is mostly obtained due to an increased shopper utilization, reduced shopping times, and cheaper routing options available.

1 Introduction

Online retailers continuously seek to offer faster delivery services to satisfy the customers’ need for instant gratification. Now, online shopping services such as Amazon Prime Now (Holsenbeck (2018)) offer one and two-hour deliveries in selected US cities.

To compete with online retailers, brick and mortar stores have associated with last-mile logistics providers offering Personal Shopper (PS) services. Such services act as intermediaries receiving online requests placed by customers using a mobile app, who request delivery of items available at affiliated brick and mortar stores. Each customer request specifies a shopping list, a delivery location and a delivery deadline. These services are increasingly popular in the delivery of groceries, since they integrate the convenience of online shopping with product availability at grocery stores.

∗Singapore Management University, (email: amuzaffera@rsm.nl)
†Rotterdam School of Management, Erasmus University (email: nagatz@rsm.nl)
‡Pontificia Universidad Católica de Chile (maklapp@ing.puc.cl)
Grocery delivery platform Instacart is now operating in 20 states across the US and has raised more than $600 million from investors. Postmates, Deliv and Google Express are other examples of this asset-light business model to grocery delivery.

Some personal shopper services offer the customer to schedule the delivery for a specific day and time in advance, and others guarantee delivery within a short lead time, e.g., 60-120 minutes. In the later, an automated dispatcher works simultaneously to the request arrival process and chooses whether or not to accept each request. Request rejections can be made implicitly by temporarily not offering the service in specific regions. All accepted requests must be purchased at specific stores and delivered to the customer on time. To perform this operation, each accepted request is assigned to a personal shopper within an available fleet. A shopper is a delivery driver who, besides performing pickup and delivery operations, also executes shopping operations, e.g., parking, walking, going through the store, purchasing and collecting the items.

The personal shopper business model is somewhat similar to on-demand meal delivery services such as Grubhub or Uber Eats. Personal shoppers, however, also execute shopping operations, and each shopping request can involve collecting items from multiple stores. Google Express, for example, has signed on over 50 merchants including Costco, Target and Walgreens.

To simplify planning, personal shopper service providers may operate a sequential delivery policy, in which shoppers serve a single customer request at a time. However, it might be operationally advantageous to combine multiple requests involving common shopping locations in one shopper trip to prorate fixed store shopping times. Unfortunately, a delivery service with such a tight delivery deadline offers limited consolidation opportunities.

In this paper, we explore different operating strategies for personal shopper services. In particular, we study the benefits of splitting the service of a customer request involving shopping at multiple stores into separate tasks served by different shoppers. Also, we study the additional consolidation opportunities that arise from splitting requests.

More shopper scheduling options are available when requests are split into tasks served by different shoppers. This may help to increase the fleet’s time and capacity utilization, which is particularly relevant when shoppers have limited capacity and tight delivery deadlines. We refer to this improvement as potential packing benefit. Figure 1 illustrates an example in which a customer request $r$ demands delivery from two stores $m_1$ and $m_2$. Two shoppers are located at $k_1$ and $k_2$. Travel and shopping times are displayed above the arcs and store nodes, respectively. Suppose that service must occur within six time units. An on-time service is infeasible when one shopper must serve the request alone. However, if the request is split into two separate store-tasks, then both shoppers can deliver to the customer by $t = 6$.

When splitting is allowed, the platform has a larger set of feasible routing options; we call it the potential routing benefit of splitting requests. The example in Figure 1 shows routing time
reductions as a single shopper must travel at least 11 time units to serve \( r \), while two shoppers spend 10 time units. Also, one could save shopping time if multiple tasks originating from a common store are assigned to a single shopper. However, consolidation opportunities are not limited to splitting requests.

In this study, we introduce the Personal Shopper Problem (PSP), which models an online shopping service dynamically receiving, accepting and serving same-day delivery requests. The objective of the PSP is to maximize the number of requests served on-time subject to a limited fleet of shoppers. All accepted requests must be assigned to shoppers and fully served before a pre-established delivery deadline.

Our main contributions are summarized as follows. (i) We identify three types of potential benefits gained by splitting customer delivery requests between different shoppers in a Personal Shopper service: packing, routing and shopping benefits. (ii) We develop an efficient solution to an online personal shopper optimization problem, which integrates request acceptance, order splitting, task to shopper assignment, and routing problems. (iii) We assess our solution quality and confirm the benefits of splitting requests with computational experiments over a different instance settings.

The remainder of this article is organized as follows. A literature review is presented Section 2. We describe the PSP in Section 3 and propose a solution for it in Section 4. Finally, Section 5 outlines results of a computational study, and we conclude in Section 6.

2 Literature Review

The Personal Shopper Problem (PSP) may be classified as a pick-up and delivery routing problem (PDP). The general PDP aims to find minimum costs routes to serve a set of transportation requests, each with known origin and destination points (Savelsbergh and Sol, 1995). The PSP generalizes the PDP, as a single transportation request can have multiple pick-up locations, i.e., stores.

When request splitting is allowed, a personal shopper operation also shares some features with multi-source fulfillment problems and multi-depot delivery problems, in which different parts of a
request can be fulfilled from different locations, see Acimovic and Graves (2014), Renaud et al. (1996), Xu et al. (2009). The option of splitting a customer request in multiple deliveries is also studied in Archetti and Speranza (2008), who formulate The Split Delivery VRP. Archetti et al. (2008) shows that splitting deliveries can be beneficial when vehicle capacity is restrictive. Similar benefits are shown for Pickup and Delivery problems in Nowak et al. (2008, 2009).

A personal shopper operation is also dynamic as new requests continuously arrive over time; see (Pillac et al., 2013) for a survey on dynamic vehicle routing problems. Recent work on dynamic routing problems has focused on same-day and on-demand delivery services, e.g., Arslan et al. (2019), Klapp et al. (2018b,a), Voccia et al. (2017). Indeed, a PSP is a SDD problem with multiple pickup locations.

The PSP is similar to the dynamic meal delivery problem (DMDP) studied in Reyes et al. (2018), Ulmer et al. (2017), Yildiz and Savelsbergh (2017). Compared to a meal delivery service, personal shopper operations work with drivers who also pick and purchase the items in the store. As such, personal shoppers spend more time at the pickup location than in a typically restaurant delivery setting. Food freshness requirements in meal delivery also impose limited dispatch-to-door times, i.e., 10 minutes or even less, and these problems are relatively more operationally constrained ending up in simpler routing problems. This difference made by the particular features of meal delivery, also creates structural differences between both problems’ solutions. Typically, shoppers in a PSP solution pick-up and carry more tasks than in DMDP. In this sense, meal delivery is relatively more operationally constrained and produces a simpler routing problem.

Recently, Steeever et al. (2019) considered a dynamic meal delivery problem with potential spitting of requests from multiple restaurants. Different from our paper, they focus on a setting with soft time windows without in-store shopping related to the request pickups. Their experiments suggest that splitting is not that beneficial in their specific setting.

3 Problem Description

In this section, we formally define the Personal Shopper Problem that models an online platform that has to dynamically decide which requests to accept and how to best serve the accepted requests. First, we describe the PSP inputs and notation and then we formulate it as a sequential decision model.

3.1 Problem Inputs, Notation, and Decisions

We consider a platform that dynamically receives \( n_R \) shopping requests throughout a service period \([0, T]\). Each request \( r \in \{1, \ldots, n_R\} \) arrives at time \( e_r \) and is composed of a set of tasks \( S_r \). Each task \( s^r \in S_r \) requires shopping a basket of items at a specific store \( m_{s^r} \) and, later, deliver it to the
request’s destination \( d_r \) before \( l_r := e_r + L \), a delivery deadline offered to the customer depending on the service lead-time \( L \), e.g., 60 or 120 minutes. Let \( M \) and \( N^R \) represent the set of relevant stores and customer locations, respectively. Furthermore, let \( S \) be the set of all tasks; i.e., \( S = \bigcup_{r \in R} S_r \).

To execute shopping and delivery operations, the personal shopper platform employs a homogeneous fleet \( K = \{1, \cdots, n_K\} \) of shoppers, each with a capacity of \( Q \) tasks. Each shopper \( k \in K \) starts and ends its daily operation at its personal location \( i \in N^K \), i.e., shopper’s \( k \) depot.

A shopper visiting a store \( m \in M \) spends a time \( c_s^p \) to pick up each task \( s \) and a fixed time \( c_m^f \) in activities such as parking, walking, going through the store, purchasing and collecting the items. Therefore, the shopping time required at store \( m \) to collect a set \( S_m \) is

\[
c(S_m) := \alpha \cdot c_m^f + (1 - \alpha) \cdot \sum_{s \in S_m} c_s^p,
\]

where \( \alpha \in [0, 1] \) is a parameter representing the relative weight of fixed shopping times.

It is possible to model the PSP on a graph where its nodes represent physical locations, i.e., customers, stores and shopper depots, while its arcs act as the movements between two locations. However, a store location should be associated to a number of different tasks and requests and a physical model makes it difficult to set task-dependent shopping times. Therefore, we instead consider a task-based graph, in which two nodes \( s_{m+}^s \) and \( s_{m-}^s \) are created for each task \( s \) representing a pickup at store \( m \), and a delivery to \( d \), respectively. As a result, we model the PSP on a graph \( G = (N, A) \), where the set of nodes \( N \) is the union of these task-based nodes over \( S \) and shopper depots \( N^K \). Figure 2 illustrates the difference between a location-based and a task-based graph with an example with one shopper, two stores and two requests. Request 1 consists of one task that involves shopping at store \( m \) and delivering to customer \( d_1 \). Request 2 consists of two tasks, requiring shopping from store \( m \) and \( m' \), respectively and delivering to customer \( d_2 \). The path displayed above the line, physically represents a shopper trip visiting stores (rectangles) and customer locations (pentagon). Below the line, that shopper trip is represented in a task-based graph, where each task shopping operation and each task delivery is represented by a node. Here, \( s_{1+}^1 \) represents shopping task 1 within request 1 from store \( m \), \( s_{1+}^2 \) represents shopping task 1 within request 2 from store \( m \), and \( s_{1+}^{2+} \) represents shopping task 2 within request 2 from store \( m' \). Analogously, \( s_{1-}^2 \), \( s_{2-}^2 \) and \( s_{1-}^1 \) represent task deliveries to customer locations.

The set of arcs \( A \) in graph \( G \) connect each task-based node \( i \in N \) to another node \( j \in N \). We define \( c_{i,j} \) as the time it takes to move from node \( i \) to node \( j \). If \( i \) and \( j \) relate to different locations, then \( c_{i,j} \) includes the travel time between both locations. Also, if \( j \) is associated with a store, then \( c_{i,j} \) also includes shopping times at that location. Appendix 7.1 presents the formal definition of arc cost.

Upon each request’s arrival, the platform decides whether or not to accept the request for service.
In this study, we assume that the platform has no prior nor probabilistic information regarding future requests. Therefore, we always accept a customer request as long as it is feasible to serve the resulting set of pending tasks after the acceptance decision. As the platform searches to feasibly serve all tasks related to the newly arriving request, it has to simultaneously (re)assign tasks to shoppers, including tasks within the same request to different shoppers, and (re)sequence planned shopper routes. If a feasible service option is found, it accepts it and continues with the operation. In the next section, we present a sequential decision model to efficiently plan these operations.

3.2 Problem Formulation

To make feasible acceptance decisions, the platform must keep track of the accepted but not yet served (active) tasks; let $S_e^a$ be the set of active tasks at time $e \in [0, T]$. Also, at time $e$, the platform has to maintain a delivery plan $\sigma_e = \{\tau_1, \cdots, \tau_{n_k}\}$ covering all tasks in $S_e^a$ and carrying for each shopper $k \in K$ a planned trip $\tau_k$. In our task-based graph $G$, each trip $\tau_k = \{i_0, i_1, \cdots, i_h\}$ is a sequence of shopping and delivery nodes, and $i_0$ indicates its shopper’s position at time $e$ or the next position if the shopper is enroute. Each trip $\tau_k$ must be feasible and, therefore, must meet capacity $Q$ and deliver all its tasks on-time. Also, each task within the trip should be shopped at the corresponding store before delivery.

At time $e$, the platform also tracks the status of each shopper $k \in K$ defined by its current position $\omega_k$, its earliest departure time from that node $\epsilon_k \geq e$, which is strictly later then $e$ if the shopper is en route, and its current load $S_k \subset S_e^a : |S_k| \leq Q$ of already collected but not yet delivered tasks. As we do not allow transfers between shoppers, these tasks have to be delivered by shopper $k$. Let $\Omega_e = \{ (\omega_k, \epsilon_k, S_k) : k \in K \}$ be the status of all shoppers at time $e \in [0, T]$.

Now, we formulate the PSP as a sequential decision model defining the system’s states, decision
epochs, actions, rewards and its objective function (see Puterman (2014)). The set of decision epochs is defined by all arrival times \( \{ e_r : r \in \{1, \ldots, n_R \} \} \). At each decision epoch \( e = e_r \), the state of the platform is \( z_e = (S^a_e, \sigma_e, \Omega_e) \). The action in state \( z_e \) is to choose a delivery plan in the collection of plans that accommodate tasks \( S_r \) and \( S^a_e \) defined by \( \mathcal{A}(z_r) \). If a feasible delivery plan exists, then the platform receives an immediate reward, i.e., \( I_e_r(z_e_r) = 1 \). Else, we set \( I_e_r(z_e_r) = 0 \) when \( |\mathcal{A}(z_r)| = \emptyset \). Define \( V_e_r \) as the cumulative reward at decision epoch \( r \) as follows:

\[
V_{e_r} = V_{e_{r-1}} + I_{e_r}(z_{e_r}).
\]  

The platform’s objective is to maximize the number of served requests by the end of the operation;

\[
\max V_T = V_{e_{n_R}}.
\]  

4 Solution Approach

Now, we describe our solution approach to the PSP problem.

Equation (3) refers to an online optimization problem. An optimal deterministic solution to it may be infeasible, since it requires perfect knowledge of future request arrivals. Instead, we focus on finding a good online policy. We regard this problem as a dynamic problem with no future information and propose the following rolling horizon framework.

4.1 Rolling Horizon Framework

To deal with the continuous arrival of requests, we use an event-based rolling horizon framework (RH) that solves a deterministic optimization problem each time a new request arrives. Based on the solution to this optimization problem, we decide whether or not to accept the new request and update the delivery plan accordingly. This approach is commonly used in the literature to model these type of systems, see Srour et al. (2016), Arslan et al. (2019). Specifically, we solve at each decision epoch a Pickup and Split Delivery Problem with Deadlines (PsDPd) that aims to identify a feasible delivery plan covering both newly arrived and active tasks. If this plan is found, then the platform accepts the new request. Instead of solving a feasibility problem, we set the objective function to minimize the total duration of all shopper trips to increase shopper’s utilization and eventually increase future acceptance capacity.

Algorithm 1 describes the RH procedure. Upon the arrival of request \( r \in R \), a routine \( Screen-theplan() \) computes all shoppers’ status and a PsDPd is solved with status \( \Omega_{e_r} \), active tasks \( S^a_{e_r} \) and newly arrived tasks \( S_r \). Function \( PsDPd \) returns either a delivery plan \( \sigma^* \) or NULL when it is infeasible to serve \( r \). If a solution is found, we increase the reward by 1 and update the delivery
plan. Otherwise, we keep the previous delivery plan and reject \( r \).

**Algorithm 1** Rolling Horizon Framework

1: **Input:**
2: \( V \leftarrow 0, \sigma = \{ \tau_k := \{ \omega_k \} : k \in K \} \)
3: **Iterations:**
4: **while** \( e_r \leq T \) **do**
5: \((S^a_{e_r}, \Omega_{e_r}) \leftarrow \text{Screen the plan}(\sigma)\)
6: \(\sigma^* \leftarrow \text{PsDPd}(\Omega_{e_r}, S^a_{e_r}, S_r)\)
7: **if** \(\sigma^* \neq \text{NULL} \) **then**
8: \( V \leftarrow V + 1 \)
9: \(\sigma \leftarrow \sigma^*\)
10: **end if**
11: **Execute(\sigma)\)
12: \( r \leftarrow r + 1 \)
13: **end while**
14: **Output:** \( V \)

The routine \( \text{Execute()} \) carries out the delivery plan until the arrival of request \( r + 1 \). This means shoppers follow the sequences of nodes as stated in their trips. A shopper departs from a node immediately when the corresponding operation on the node is over. A shopper can be located either somewhere on his or her current trip (i.e., busy) or at the final destination of the last completed trip (i.e., idle). All shoppers return to their depots after the service period is over and all assigned tasks are served.

### 4.2 Deterministic Pickup and Split Delivery Problem with Deadlines

At each decision epoch, we solve a deterministic routing problem where customer requests can be split and served by multiple drivers in parallel. We refer to this as the Pickup and Split Delivery Problem with Deadlines (PsDPd)

Let \( \text{PsDPd}(\Omega_{e_r}, S^a_{e_r}, S_r) \) be the problem solved with the arrival of request \( r \). For the sake of convenience, let’s assume that \( S^a = S^a_{e_r} \cup S_r \). The PsDPd seeks a set of feasible shopper trips serving all tasks in \( S^a \) while minimizing the sum of all trip durations.

In a task-based graph, the PsDPd can be formulated as a Pickup and Delivery Problem with Time Windows (PDPTW). However, we may exploit the fact that we may be able to consolidate shopping operations of multiple requests at a single store. Observation 1 uses this idea to reduce the search space.

**Observation 1.** Let \( \tau = \{i_0^-, i_1^+, X, i_1^-, \cdots \} \) and \( \tau' = \{i_0^-, i_1^+, X', i_1^-, \cdots \} \) be trips such that both have the same node sequences except partial sequences \( X \) and \( X' \). Also, let \( X \) and \( X' \) be the two
different permutations of a set of nodes such that each element of the set maps to same location. Then, two trips \( \tau \) and \( \tau' \) are identical in terms of trip duration.

4.2.1 Exact Approach

This section describes an exact approach to solve the PsDPd that partitions all active tasks among the fleet of shoppers and solves the individual shopper routing problems to optimality. Let \( p = \{ P_1, \ldots, P_n \} \) be a partition of the set of active tasks \( S^a \), where \( P_k \subset S^a \) is the set of tasks assigned to shopper \( k \). Let \( \mathbb{P} \) be the collection of all feasible partitions. The partition \( p \) is feasible if each shopper \( k \in K \) has a feasible trip covering \( P_k \).

To find an optimal delivery plan, we enumerate all feasible partitions \( \mathbb{P} \) and determine the optimal delivery plan for each one as follows. For a given set of shopper tasks \( P_k \), we can find the trip for shopper \( k \) that serves all tasks while minimizing the total travel time by solving a Hamiltonian Path Problem with Precedence Constraints and Deadlines. We can certify that the set of optimal trips for all shoppers in a partition \( p \) forms an optimal delivery plan for a given partition \( p \). To identify which partition minimizes the total service time, we explore all partitions sequentially. To do so, we store the best partition (delivery plan) and the associated objective value as an incumbent plan as an upper-bound value. For each unexplored partition, we start solving Hamiltonian path problems for each shopper. If a lower bound to the minimum feasible duration of all shoppers’ routes exceeds the incumbent, then we stop and discard the partition. If the solution takes less time than the current best solution, we update the incumbent plan. The approach terminates when all partitions are checked. The final incumbent solution is the delivery plan that minimizes the total service time.

Each Hamiltonian Path Problem is solved by a forward labeling algorithm, similar to the one proposed by Tilk and Irnich (2016). In this method, partial shopper trips are constructed and recursively extended via a resource extension function. Let \( x_i \) be a partial trip from a source node; e.g., shopper’s \( k \) current location, to a node \( i \in P_k \) with label \( (X, c, i) \), where \( X \subset P_k \) is the ordered subset of visited nodes in the partial trip, \( c \) is the trip’s duration and \( i \) is the last node in \( x_i \). The initial label for shopper \( k \) is \( (X = \{\omega_k\}, 0, \omega_k) \). A label extends to node \( j \in P_k \) and produces a new label \( (X \cup \{j\}, c, j) \) with the following REFs: \( c_j = REF_{ij}(c_i) := c_i + t_{ij} \). An extended (partial) trip can be infeasible, feasible but dominated, or non-dominated. We eliminate all trips that are dominated.

Algorithm 2 describes the forward labeling algorithm for shopper \( k \) over tasks \( P_k \). Let \( X_l \) be collection of partial trips with length \( l = |X| - 1 \). Subroutine \( \text{FeasibilityCheck()} \) evaluates whether or not an extension of the partial path to node \( j \) meets delivery deadlines. It is feasible to extend a partial trip if it is possible to reach node \( j \) before the deadline. Routine \( \text{BoundingCheck()} \) verifies if the total duration of the partial plan for Partition \( P_k \) exceeds the incumbent’s solution, and if so, stops the algorithm and prunes the partition.
Algorithm 2 Forward labeling algorithm

1: Set $X_0 = \{X = \{\omega_k\}, 0, \omega_k\}$
2: for $l = 1 \cdots |P_k| - 2$ do
3:   for $x \in X_l$ do
4:     for $j \in P_k, j \notin X$ do $c_j = REF_{ij}(c_i)$
5:      if FeasibilityCheck$(c_j)$ then
6:        if BoundingCheck$(c_j)$ then
7:          Add $x_j$ to $X_{l+1}$
8:        end if
9:     end if
10: end for
11: end for
12: DominanceRule($X_{l+1}$),
13: end for
14: Find a path $x$ with label $(P_k, c, \cdot)$ such that $c$ is minimal.

The dominance rule eliminates partial trips according to the following rule:

**Definition 1.** Let $x_i$ and $x_i'$ be two partial trips for the same shopper with labels $(X, c, i)$ and $(X', c', i)$ with common last node $i$. Partial trip $x_i$ dominates $x_i'$ if $X' \subset X$ and $c < c'$.

Furthermore, **Observation 1** allows us to eliminate redundant partial paths by forcing a lexicographical order for nodes corresponding to the same location without loss of generality.

### 4.2.2 A Heuristic for the PsDPd-Planmaker()

Solving the PsDPd exactly is hard in large instances. Therefore, we propose an adaptive large neighbourhood search (ALNS) heuristic called **Planmaker()** similar to the one proposed by Pisinger and Ropke (2010).

When a request $r$ arrives at time $e_r$, we try inserting all its tasks into the delivery plan $\sigma$. For each unassigned task $s \in S_r$, the procedure determines the cheapest insertion for its pickup ($s^+$) and delivery tasks ($s^-$) simultaneously examining all possible options in each shopper trip. Then, the cheapest unassigned task is inserted into the plan. This process continues until there is no unassigned task left to insert (or when it is not feasible to do so).

Later, we run ALNS over neighborhoods defined by removal and repair operators. Removal operators destroy the solution to a predefined level by removing certain tasks. Repair operators reinsert the removed tasks into the delivery plan.

**Destroy Operators.** For a number of $n^a_s$ tasks and $n^a_r$ requests in the incumbent solution, we set a removal ratio $\rho \in [0, 1]$ and, depending on the following five operators we remove $q_s = \rho \cdot n^a_s$ tasks or $q_r = \rho \cdot n^a_r$ requests.
- **Random task removal.** Randomly remove \( q_s \) tasks.

- **Random request removal.** Randomly remove \( q_r \) requests.

- **Most time-consuming task removal.** Remove the \( q_s \) most time-consuming tasks from the current solution. The time-consumption of a task is defined as the time savings generated if the task is ejected from the route.

- **Most time-consuming request removal.** Remove the \( q_r \) most time-consuming requests.

- **Shaw Removal.** Randomly remove a task \( s \), and then also remove the \( q_s - 1 \) closest tasks to \( s \) in terms of the euclidean distance.

**Repair Operators** Once tasks are removed, they are sequentially inserted back into the delivery plan based on the following operators:

- **Cheapest insertion:** Insertion sequence goes in increasing order of insertion cost.

- **Non-split insertion:** All removed tasks from the same request are inserted into the tour of one shopper.

- **Regret insertion.** Tasks are inserted in decreasing order of regret, defined as the difference between the cheapest and the second cheapest cost.

Finally, we choose the repair operators according to weights, which are updated based on the success of each operator.

## 5 Computational Study

In this section, we present an extensive set of computational experiments to assess the quality of our solution approach and explore the potential benefits of different operational strategies for the Personal Shopper Service. Next, we describe the experimental setting and instances for the base-case experiments.

### 5.1 Experimental Setup

All our experiments use a circular service area with a 10km radius in which the request destinations are uniformly distributed. We consider five stores in the service region: one in the center, and four located on an inner circle of radius \( r_s = 5 \)km at equal intervals (see Figure 3).

The base case experiment considers instances with three shoppers and 80 request arrivals within a ten hour service period. We consider exponential request inter-arrival times and, thus, realizations are uniformly distributed within the service period. However, this probabilistic information
is assumed unknown to the platform. Also, we assume that each request requires to shop three tasks, each at a different store randomly selected from the five stores available. To simplify the interpretation of the results, we also assume that each shopping task consists of the same number of items with the same weight and volume, and that each shopper has capacity $Q = 10$ tasks. Shoppers travel at a speed of 30 km/h. We set $\alpha$ to 0.9 and set all shopping time values equal to ten minutes, meaning that a shopper spends nine minutes for each store visit, and one minute to pick up a task within a store. Table 1 summarizes all parameter values used in the base case experiment.

<table>
<thead>
<tr>
<th>Table 1: Base Case Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of shoppers</td>
</tr>
<tr>
<td>Shopper speed</td>
</tr>
<tr>
<td>Shopping time coefficients:</td>
</tr>
<tr>
<td>$c_f$</td>
</tr>
<tr>
<td>$c_p$</td>
</tr>
<tr>
<td>$\alpha$</td>
</tr>
<tr>
<td>Deadline</td>
</tr>
<tr>
<td>Shopper capacity</td>
</tr>
</tbody>
</table>

5.2 PsDPd Heuristic Validation

Now, we evaluate the performance of our Planmaker() heuristic compared to the exact solution of the PsDPd. We collect an instance set for the PsDPd by executing the RH framework over the online problem as described in Section 4.2. To solve the deterministic snapshot problems to optimality, we consider a small instance with 40 request arrivals and two shoppers in this validation process. We draw five instances for each sub-problem with between 8 and 15 active tasks.

Table 2 presents the average percentage difference in total service time between the heuristic and the exact approach, and the number of times an optimal solution is found by the heuristic. However,
more importantly, we show the number of times that PlanMaker() fails to find a feasible solution while the exact approach does find one.

Table 2: Comparison between exact and heuristic solutions for the PsDPd

<table>
<thead>
<tr>
<th>n_s</th>
<th>Av. Opt. gap (%)</th>
<th># Optimal</th>
<th>△ Feasibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1.3</td>
<td>4/5</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1.5</td>
<td>4/5</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>4.0</td>
<td>3/5</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0.8</td>
<td>4/5</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>2.1</td>
<td>3/5</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>2.8</td>
<td>4/5</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>3.7</td>
<td>1/5</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>3.1(^1)</td>
<td>0/5</td>
<td>1</td>
</tr>
</tbody>
</table>

The results show that the heuristic performs well with an average optimality gap of less than 4.0%. Although it does not always find the optimal solution, PlanMaker() identifies a feasible solution and, thus, makes a correct acceptance decision in all but one case.

5.3 Operating Policies

For each simulation, we test three different policies computing performance metrics in each one:

One by One (1b1): Each shopper serves one single customer request at a time and cannot start serving a new request before delivering the previous one.

Consolidation (C): A shopper can simultaneously serve multiple requests, e.g., shopping for one request and then delivering another. In this policy, all tasks of a single request are served by one single shopper.

Consolidation & Splitting (C&S): A request can be split into different store tasks that can be served by multiple shoppers in parallel. As in the consolidation strategy, shoppers can simultaneously serve tasks of multiple requests.

5.4 Base Case Results

Table 3 presents the results for the different policies averaged over 99 random scenarios of request arrivals. We report the following performance indicators: (i) Served requests: the number of served requests as a percentage of the total number of requests. (ii) Requests split: the number of split requests as a percentage of total served requests. (iii) Time per request: The total time worked by all shoppers divided by the number of requests served. We further break this value down in two parts:

\(^1\)Average over four instances. Our heuristic did not find a feasible solution for one instance
Shopping time and Travel time per request. (iv) Click to door (CtD): the average time between a request’s arrival and the delivery of its latest task. (v) Delivery interval: the average time between the delivery of the first and the last task of a specific request. By definition, the interval is equal to zero for requests that are not split. (vi) Number of locations visited per request: The total number of visited locations (stores and customers) over all shoppers divided by the number of requests served.

Table 3: Average base case results, 80 Requests, L: 90 minutes, 3 tasks per request, $n_K$: 3 shoppers

<table>
<thead>
<tr>
<th></th>
<th>1b1</th>
<th>$C$</th>
<th>C&amp;S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Served requests (%)</td>
<td>45.8</td>
<td>77.3</td>
<td>88.1</td>
</tr>
<tr>
<td>Request split. (%)</td>
<td>0</td>
<td>0</td>
<td>69.2</td>
</tr>
<tr>
<td>Delivery interval (min.)</td>
<td>0</td>
<td>0</td>
<td>23.6</td>
</tr>
<tr>
<td>Time per req (min.)</td>
<td>51.2</td>
<td>28.9</td>
<td>25.1</td>
</tr>
<tr>
<td>Shopping time per req (min.)</td>
<td>30.0</td>
<td>15.0</td>
<td>10.3</td>
</tr>
<tr>
<td>Travel time per req (min.)</td>
<td>21.2</td>
<td>13.9</td>
<td>14.8</td>
</tr>
<tr>
<td># locations visited per req.</td>
<td>4.0</td>
<td>2.3</td>
<td>2.7</td>
</tr>
<tr>
<td>CtD (min.)</td>
<td>77.1</td>
<td>78.2</td>
<td>77.3</td>
</tr>
</tbody>
</table>

Our results show that substantial performance improvements can be obtained by consolidating requests. Comparing the consolidation policy $C$ to the 1b1 policy, the average numbers of served requests increases from 45.8% to 77.3%. This improvement is associated with a reduction in the fulfillment time per request from 51.2 to 28.9 minutes, both in terms of shopping and travel times.

An additional performance increase of 15% is possible when request splitting is allowed, i.e., C&S. Splitting enables more consolidation opportunities by reducing granularity and increasing the packing benefit. Moreover, we also observe a reduction in the service time per request, which indicates extra shopping benefits. Conversely, there is a slight increase in travel time per request, i.e., from 2.3 to 2.7. This could relate to the fact that splitting means that a request destination is visited multiple times. Overall, the travel time per location visited decreases from 6.0 (13.9/2.3) to 5.4 (14.8/2.7) which indicates that there are routing benefits.

When we allow splitting, we see that 69.2% of the accepted requests are split and thus served by more than one shopper. The average time between the first partial delivery and the final partial delivery, i.e., the delivery interval, is 23.5 minutes while the click-to-door (CtD) time is between 77 and 78 minutes for each of the different policies². This suggests that the first partial deliveries occur earlier than the non-split deliveries.

Figure 4 presents an histogram of the differences in the number of requests served for policies C&S and C over all simulated scenarios. We see that the number of accepted customer is no smaller for C&S than C in all but two instances.

²See Appendix 7.2 for more detailed analysis
5.5 Impact of the Number of Stores per Request

Now, we vary the number of tasks per request between two and five. This also varies the number of different stores involved per request. To allow for comparison, the same stream of requests is used in each of the different scenarios. The only difference is the number of tasks per request. To keep the average number of tasks per shopper the same in the different scenarios, we adapt the number of shoppers accordingly.

Our results are summarized in Figure 5. Figure 5a presents the percentage of served requests for the different numbers of stores (tasks) per request and policy. For the consolidation policy \((C)\), we see that the performance deteriorates with more tasks per request. One potential reason for this observation is that it is more difficult to fit larger requests into the shoppers’ time schedules. Moreover, it is also more difficult to combine multiple larger request in one shopper to exploit the economies of scale in shopping. However, the consolidating policy \((C)\) consistently outperforms the simple \(1\text{b}1\) policy, even for requests that involve five stores.

In contrast to policy \((C)\), we see that the performance of the \(C\&S\) policy improves with the number of stores per request. The reason for this is that this policy assigns tasks (instead of requests) to shoppers. This fact facilitates that shoppers can collect more tasks at store visits and also traverse fewer stores before delivery locations with policy \(C\&S\) comparing to \(C\).

Figure 5b reports the average click-to-door (CtD) times for the different numbers of tasks per
request. As expected, the CtD time increases with the number of tasks per request. This is because each store visit involves travel time to the store and shopping time in the store. We observe that the policies without splitting are more sensitive to variations in the number of stores per request. This implies that splitting also reduces the fulfillment time per customer.

The box plots in Figure 5c and Figure 5d provide more insight into the solutions for the C&S policy. Figure 5c shows that the number of requests with at least one split increases with the number of tasks per request. Figure 5d show the same for the number of requests with at least two splits. These results are intuitive as the number of splitting options increases with the number of tasks. We see that in the instances with five tasks per request almost all requests are split at least once.

**Figure 5: Impact of Request Size, 80 Requests, α : 0.9, L: 90 minutes**

5.6 Impact of Shopping Economies of Scale $\alpha$

In this section, we study the impact of the relative weight $\alpha \in [0, 1]$ of the fixed part of the shopping time, i.e., ‘setup time’, on the performance of the different proposed policies. A value of $\alpha = 0$ indicates that there is no setup time and the shopping time at the stores is directly proportional to the number of tasks collected. In this case, there are no economies of scale in the shopping
operations. Conversely, a value of $\alpha = 1$ represents a situation in which there is only a setup time and the total shopping times at the store are independent of the number of tasks picked up. In our base experiments, we assume the fixed and the variable part of the shopping time to be equal, so that total shopping time is independent of $\alpha$ when no consolidation occurs.

Figure 6 presents the performance for the different values of $\alpha$ for all policies. Figure 6a shows that for policies with consolidation ($C$ and $C&S$) the percentage of served requests increases with $\alpha$. This is intuitive as the value of consolidating increases with a higher value of $\alpha$. As expected, the economies of scale do not effect the performance of the $1b1$ policy as it does not allow consolidations. Comparing $C$ and $C&S$, we see that the benefits of splitting, i.e., shopping benefits, increase with $\alpha$. For values $\alpha < 0.5$ there are no benefits as the savings in the shopping time do not offset the additional travel time associated with splitting requests.

Figure 6b reports the average shopping time per served request. This illustrates the realized economies of scale in the shopping activities. Without consolidation, each 3-task request involves 30 minutes of shopping. With consolidation ($C$ and $C&S$), the average shopping time per request decreases as $\alpha$ increases. By simultaneously shopping multiple tasks in a store, it is possible to reduce the shopping time per task; this reduction is higher when requests can be split.

Similarly, we observe in Figure 6c and 6d that for policies $C$ and $C&S$ the average travel time and number of locations visited per request reduce as $\alpha$ increases. If there are more incentives for consolidation, then the platform will efficiently pack tasks to reduce the number of nodes visited per request and, thus, travel time. We further observe that the average difference in travel time per request between policies $C$ and $C&S$ reduce as $\alpha$ increases.

5.7 Impact of the Delivery Deadline on the Packing Benefits

In the previous section, we saw that the consolidation of shopping tasks is an important driver for the benefits of splitting. In this section and the next section, we focus specifically on the packing and routing benefits by setting $\alpha = 0$, which means that there are no economies of scale in the shopping activities.

Figure 7 presents the average number of requests served for policies $C$ and $C&S$ for different delivery deadlines ranging from 45 to 60 minutes. We see that the performance of both policies is similar with a 60 minute deadline. However, we see that the policy that allows splits outperforms the consolidation policy when the deadlines become shorter and the instances become more time constrained. That is, the performance of policy $C$ deteriorates significantly with a reduction of the delivery deadline. Request splitting allows for more planning flexibility by creating smaller tasks that can be served in parallel which help to improve the capacity utilization, i.e., the packing benefits. This suggests that request splitting is especially beneficial in systems with short delivery deadlines and tight time constraints.
Figure 6: Impact of Shopping Consolidation Factor, 80 Requests, L: 90 minutes, $n_K$:3

(a) Served requests

(b) Shopping time per request

(c) Travel time per request

(d) # of locations visited per request

5.8 Routing Benefits

In this section, we focus specifically on the routing benefits of request splitting. Therefore, we consider a setting without shopping benefits ($\alpha = 0$) and no packing benefits, i.e., sufficient capacity. In particular, we increase the number of shoppers so that policy $C$ can serve all requests.

Figure 8 presents box-plots for the average difference between the travel time per request in policy $C$ and $C&S$. We choose two different delivery deadlines (90 and 120 min.) and two different fleet sizes, with 15 and 20 shoppers, respectively. The longer deadlines in combination with the large fleet ensures that all requests can be served in both policies for all the instances.

We see that for all parameter settings, request splitting provides additional routing benefits in most of the instances. Furthermore, we observe that this benefit increases when the fleet size is smaller, or the lead-time is shorter; i.e., the resources are more restricted for a request. The primary reason behind it is that the number of feasible solutions available for policy $C$ are sensitive to
the time and fleet restrictions. However, splitting requests allows to partially reassign node visits between shoppers so that the overall delivery plan has more options to minimize travel time.

These results suggest that there are benefits to splitting even in settings with sufficient capacity and time to serve all requests.

6 Concluding Remarks

The paper focuses on a personal shopper service that provides same-day delivery from brick and mortar stores. Customers place orders throughout the day requesting delivery from one or more stores. The service provider decides whether or not to accept a request and on how to assign shoppers to request in order to maximize the number of served customer requests. We model the problem as a sequential decision problem and present a rolling horizon approach to solve it. We propose an exact and a heuristic approach to solve the pickup and delivery problems at each optimization interval.

We have conducted an extensive computational study to compare different operating strategies, in terms of number of served request and total travel time. Our results provide the following insights. Consolidating requests increases the number of served requests as compared to a strategy in which all requests are served one by one. Moreover, splitting requests between different shoppers can further enhance the performance of the system. The benefits of splitting increase when the system is more constrained or when there are more economies of scale in shopping.

As this is one of the first papers that studies the personal shopper business model, there are many avenues for future research. One natural extension is to consider a setting in which we have
probabilistic information on the customer requests. Here, it would be possible to strategically reserve some delivery options for more profitable future customer requests. Another interesting research avenue involves incorporating both splits and transfers. This means allowing transfers of shopped tasks between shoppers to consolidate delivery operations.

References


_ , Martin WP Savelsbergh, and M Grazia Speranza, “To split or not to split: That is the question,” Transportation Research Part E: Logistics and Transportation Review, 2008, 44 (1), 114–123.


7 Appendix

7.1 Arc Costs in the Task-Based Graph

In the task-based graph, the cost of an arc consists of multiple components (e.g., parking time, task picking time, and, travel duration between different locations.), as this value depends on the source and sink nodes of the arc.

In equation 4, we formally define arc costs on the task-based graph, defined in Section 3. Let \( t_{ij} \) be the average travel time from node \( i \) and \( j \). Notice that this value is positive only if these \( i \) and \( j \) are not located at the same physical point; define the logical operator \( i \neq j \) that checks whether \( i \) and \( j \) are two different physical location.

\[
e_{ij} := \mathbb{1}_{i \neq j} t_{ij} + \mathbb{1}_{j \in S^+} \left( c_p^j + \mathbb{1}_{i \neq j} c_f^j \right),
\]

where \( \mathbb{1}_{\cdot} \) is an indicator function that takes value one if the statement in \( \cdot \) is true.

7.2 Distributions in Base Case for Policy C&S

We provide additional results for the C&S policy in the base experiments. In Table 4, we present the empirical distribution of click-to-door times and delivery intervals. Click-to-door times are displayed for both requests split and non-split in the operation. Delivery intervals are only computed for requests, which were split during service.

Table 4: Empirical Distribution of click-to-door time and delivery interval for the C&S policy.

<table>
<thead>
<tr>
<th></th>
<th>click to door (CtD)</th>
<th>delivery interval</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>non-split</td>
<td>split</td>
</tr>
<tr>
<td>minimum</td>
<td>15.0</td>
<td>37.8</td>
</tr>
<tr>
<td>Q1</td>
<td>71.4</td>
<td>72.9</td>
</tr>
<tr>
<td>Q2</td>
<td>81.7</td>
<td>82.2</td>
</tr>
<tr>
<td>Q3</td>
<td>87.2</td>
<td>86.8</td>
</tr>
<tr>
<td>maximum</td>
<td>89.9</td>
<td>89.9</td>
</tr>
<tr>
<td>average</td>
<td>77.2</td>
<td>78.1</td>
</tr>
</tbody>
</table>

These results suggest that click to door times do not significantly change when request splitting is allowed. Table 4 also shows that delivery intervals range between a couple of seconds to 85.7 minutes. Nonetheless, the average delivery interval is 23.4 minutes and three out of four cases are less than 34.9 minutes.