Operational Strategies for On-demand Personal Shopper Services

Alp Arslan[*], Niels Agatz[†], and Mathias Klapp[‡]

April 29, 2020

Abstract

Inspired by several recent startups, we study an on-demand delivery service that lets customers shop online for products from a number of brick and mortar stores. The customer orders are fulfilled by a fleet of personal shoppers who are responsible for both the shopping of orders at the stores and the delivery of these to customer locations. The operation of such a service requires to dynamically manage new requests, coordinate a fleet of shoppers, schedule shopping operations at stores, and execute deliveries to customers on time.

Our work presents three operational strategies, each requiring different levels of shopper flexibility and implementation complexity. We quantify the performance of each strategy in a vast family of computational experiments. Also, the performance of this on-demand shopping service is compared to a setting in which customers travel to stores to shop themselves. Our numerical experiments show that there are significant savings in resources spent in shopping (up to 55.2%) when this activity is outsourced.

1 Introduction

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

Moreover, we see the rise of online platforms that allow customers to receive expedited deliveries from local brick and mortar stores. These platforms aggregate the assortment of different affiliated stores to provide a convenient 'one-stop shopping' experience.

These services are increasingly popular in the delivery of groceries, since they integrate the convenience of online shopping with product availability at grocery stores. Grocery delivery platform Instacart is now operating in all major cities across the US and Canada and has raised more than

[*]Singapore Management University, (email: muzalparslan@gmail.com)
[†]Rotterdam School of Management, Erasmus University (email: nagatz@rsm.nl)
[‡]Pontificia Universidad Católica de Chile (maklapp@ing.puc.cl)

$600 million from investors (Sampath, 2020, Rey, 2018). Postmates, Deliv and Google Express (Now part of Google shopping) are other examples of this *asset-light* business model to grocery delivery.

Most personal shopper services offer same-day on-demand delivery within a short lead time, *e.g.*, 60-120 minutes. Here, an automated dispatcher dynamically manages requests requiring purchases at specific stores and delivery to the customer on time. To perform this operation, each request is assigned to a *personal shopper* within an available fleet.

What differentiates this service from other delivery operations such as meal delivery, is that personal shoppers deliver not only the goods but also execute shopping operations, which involves picking the items in the store. Operational objectives are also different since meal delivery planners aim to minimize dispatch-to-door times to satisfy food freshness requirements, which limits flexibility in route planning. Furthermore, the personal shopper may have to visit multiple retail stores to serve a single customer.

In this paper, we introduce the Personal Shopper Problem (PSP), which models an online shopping service dynamically receiving, and serving on-demand shopping and delivery requests. The objective of the PSP is to minimize the total time to serve all delivery requests. All requests must be assigned to shoppers and fully served before a known delivery deadline.

To simplify planning, personal shopper service providers may operate a sequential delivery strategy, in which each shopper serves a single customer request at a time. However, it may be possible to reduce shopping and travel times by combining multiple requests in one trip. This improvement is especially true if the different requests share the same shopping locations, or if delivery locations are close. Unfortunately, a delivery service with tight delivery deadlines offers limited consolidation opportunities. Therefore, we explore different operational strategies for personal shopper services. In particular, we study the benefits of splitting a customer request that involves shopping at multiple stores into separate tasks that are served by different shoppers. Also, we study the additional consolidation opportunities that arise from splitting requests.

Our main contributions are summarized as follows. (i) We are the first to study a new service model in which personal shoppers shop for groceries at local stores to be delivered to the customer. (ii) We define the Personal Shopper Problem model and present a rolling horizon framework, which dynamically solves a static snapshot version of the problem. Moreover, we present three operational strategies, each involving a different setting of request consolidation and service splitting. (iii) We assess our solution approach and the performance of different operational strategies with a set of computational experiments over different instance settings. (iv) We also numerically study the effectiveness of the personal shopper service model compared to a setting in which customers travel to the stores and shop for themselves; *i.e.*, a Do it Yourself (DIY) benchmark. Even for our simplest operational strategy, the system's total travel time per request reduces by 12.9% on average, when the shopping activity is outsourced. These time savings significantly increase when request

consolidation and splitting strategies are implemented. (v) Finally, we observe that the number of shoppers required to cover all requests reduces as request consolidation and request splitting strategies are implemented.

The remainder of this paper is organized as follows. Section 2 provides a review of the related literature. We present the problem setting, a decision framework and operational strategies in Section 3 and 4. Section 5 presents solution methods for the snapshot optimization problem. Section 6 outlines the results of our computational study. Finally, section 7 provides conclusions and future research opportunities.

2 Literature Review

Conceptually, the Personal Shopper Problem (PSP) can be characterized as a pick-up and delivery problem (PDP), which involves designing cost-efficient routes that serve a set of transport requests, each with a specific origin and destination (Savelsbergh and Sol, 1995). The PSP extends the PDP, as one single transportation request may require pickups at multiple locations. Moreover, while the 'pickup time' is often negligible as compared to the travel time in the PDP, it is significant for the PSP and dependents on the number of simultaneous pickups per origin location.

One important feature of the personal shopper operation is that is has dynamic request realizations over time. Recently, 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, 2019), Voccia et al. (2017), Ulmer et al. (2018).

In a dynamic environment, customers place requests dynamically over time while the pickup and delivery operation is being executed; see (Pillac et al., 2013) for a survey on dynamic vehicle routing problems. An efficient decision support tool in such a setting should not only decide the sequence of the pickups and deliveries, but also when to dispatch each shopper to enhance order consolidation. Also, it has to dynamically adapt delivery plans to newly revealed requests considering that the system may benefit from consolidation opportunities with pending visits. For example, assigning a request with pickup and delivery locations close to already committed visits may only lead to a marginal increase in route duration.

Similar consolidation problems have also been studied in the context of order picking operations in warehouses (De Koster et al., 2007, Van Gils et al., 2018). Here, this is often referred to as 'order batching' (Henn et al., 2012, Yu and De Koster, 2009). The main difference is that while order pickers have fixed start and end points and move around in simple and structured warehouse lay-outs, the personal shopper has to navigate between different stores and customer locations. This means that the underlying routing problems are less challenging than in the personal shopper context.

The concept of 'zone picking' (Jane and Laih, 2005) for warehouse order pickers is also related

to the PSP. In zone picking, each picker is responsible for a dedicated picking zone, while each order may be associated to multiple zones and split between multiple pickers. The PSP has some similarities with zone picking and may be seen as a dynamic version in which zones are assigned in real time.

When splitting requests is allowed, the personal shopper operation also shares some features of the Split Delivery Vehicle Routing Problem in which it is allowed to split the service of each customer into multiple visits (Archetti and Speranza, 2008). The work in Archetti et al. (2008) shows that total vehicle travel may be reduced if customer services can be split into multiple visits; this is particularly important if vehicle capacity is restrictive. Similar benefits are presented for Pickup and Delivery problems in Nowak et al. (2008, 2009).

The PSP also has some similarities with on-demand meal delivery (Reyes et al., 2018, Steever et al., 2019, Ulmer et al., 2017, Yildiz and Savelsbergh, 2017). However, freshness requirements severally limit consolidation opportunities in the meal delivery context. Steever et al. (2019) consider 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 Personal Shopper Problem

In this section, we define a Personal Shopper service operation. We first describe how customers place requests and how these are shopped and delivered by personal shoppers operated by an automated decision-maker, *i.e.*, the platform, which dynamically assigns orders to shoppers and plans shopper trips.

3.1 Customer requests and shopping dynamics

A personal shopper service offers an online platform, where customers can shop items from a set M of local stores. These items are shopped and delivered to the customers' addresses within a time period L, *i.e.*, a maximum click-to-door time defining the service level. Requests arrive dynamically throughout an pre-established operating day, *i.e.*, T time units. Let $R := \{1, \ldots, n_R\}$ be the set of all customer requests made during the day. Each individual request $r \in R$ is placed by a customer at a specific time $e_r \in (0, T)$; it consists of a set S_r of shopping tasks to be shopped at one or more stores and delivered to a customer location d_r before time $l_r := e_r + L$. Each task $s^r \in S_r$ involves purchasing a number of items at a specific store $m_{s^r} \in M$.

To shop and deliver all requests, the personal shopper service coordinates a set $K = \{1, \cdots, n_K\}$ of personal *shoppers*, who are homogeneous delivery employees. Each shopper $k \in K$ executes

shopping and delivery tasks as decided by the platform, carries at most Q tasks at a time and starts and ends each day at a base location i_k.

When a shopper visits a store $m \in M$ to purchase the requested items, they spend a fixed 'set-up' time and a variable time that depends on the number of items that need to be shopped. In particular, they spend p_s time units per each shopping task s and a fixed time f_m in activities such as parking, walking, going through the store, purchasing and collecting the items.

The advantages of combining different tasks in a single shopping trip are related to a reduction in the fixed time component. That is, the shopping time required at store m to collect a set S of tasks is

$$c(S) := f_m + \sum_{s \in S} p_s. \tag{1}$$

Also, we assume a known travel time function between all pairs of nodes i, j within a set N describing all relevant locations, including customer nodes, stores and shopper bases.

3.2 Dynamic decisions

The online platform manages real-time operations of the personal shopper service. At any time $t \leq T$, the personal shopper platform requires a feasible *delivery plan* σ to coordinate the activities of each shopper in the fleet. Such a plan must specify the sequence of shopping and delivery tasks scheduled to be executed by each shopper from time t onward. To be feasible, σ must cover all active (*i.e.*, pending and yet unattended) tasks before their respective deadlines, comply with shopper capacity, and satisfy precedence rules between the shopping and delivery tasks within each request. When a new customer request is placed online, the plan must also be updated. To do that, one has to simultaneously (re)assign tasks to shoppers and (re)sequence planned shopper routes.

In this study, we assume that the platform has no prior nor probabilistic information regarding future requests. This is a reasonable assumption for platforms that only recently started operations. Also, we assume that we operate with a large enough fleet to cover all customer requests on time. The objective is to design a sequential decision model to cost-effectively plan the personal shopper service operation.

3.3 Rolling horizon framework

To execute dynamic decisions, we use an event-based rolling horizon framework. This approach solves a deterministic optimization problem each time a new request arrives. Based on the solution to this optimization problem, we decide to update the delivery plan accordingly. This approach

is commonly used in the literature to model similar dynamic delivery operations, see Srour et al. (2016), Arslan et al. (2019).

Figure 1 presents the main structure of our rolling horizon framework. At each time $t \leq T$ the rolling horizon framework carries the system's **state** (S^a, σ), which tracks a set S^a of active tasks and the current delivery plan $\sigma = \{\tau_1, \cdots, \tau_{n_K}\}$ covering all tasks in S^a and carrying a planned trip τ_k for each shopper $k \in K$. A trip τ_k is a sequence of locations in N, each with potentially multiple shopping or delivery tasks assigned meeting precedence requirements. Each trip determines its current load $S_k(\tau_k)$ of already collected active tasks, the current location $\omega_k(\tau_k) \in N$ of each shopper k, and its earliest departure time from that node $e_k(\tau_k) \geq t$; this time is strictly later than t if the shopper is en route. We assume that no transfers are allowed between shoppers, *i.e.*, all tasks in tasks S_k have to be delivered by shopper k.

When a new request r arrives, the delivery plan σ is revised and updated to include the shopping and delivery requirements within r. To perform this update, we solve a static *Shopper Routing Problem* (SRP) to identify a feasible delivery plan covering both newly arrived and active tasks. It must assign all new tasks in r to shoppers, (re)assigns active tasks and redefine shopper visiting sequences in each shopper trip.

The objective function of the SRP is set to minimize the total shopper time consumed; this choice increases the time utilization of shoppers.

We refer the reader to Section 5 for a detailed description of our solution approaches for the SRP.

Also, the set of feasible delivery plans admitted in the SRP depend directly on the type of operational strategy implemented by the platform; we refer the reader to Section 4 for an overview of all operational strategies considered.

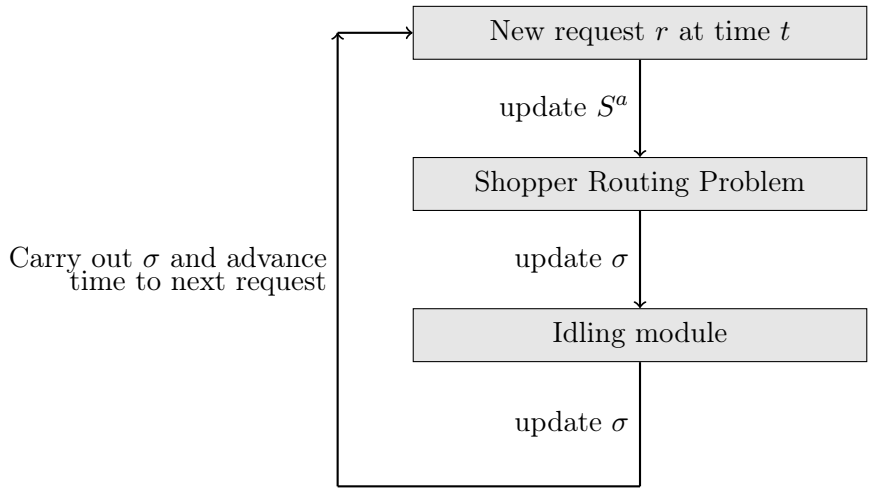
Furthermore, the platform dynamically decides if idle shoppers must be relocated calling an *Idling module* after the SRP is solved. It may be convenient for the system to relocate idle shoppers to reduce future travel times to stores. The resulting delivery plan is executed and time is advanced to the next decision epoch.

4 Operational Strategies

We study the performance of three different operational strategies, which differ in the possibility to consolidate requests in one shopper trip and split the service of a request involving shopping tasks at multiple stores into different shoppers.

> One by One (1*b*1): This strategy assumes that each shopper serves one customer request at a time, *i.e.*, a shopper cannot start shopping for a new request before delivering the previous one. It emulates the pick-by-order strategy in warehouse order picking operations.

Figure 1: Rolling horizon framework



Consolidation (C): This strategy allows to combine multiple shopping tasks from different requests before delivering these in each shopper trip. Nonetheless, all tasks of a single request must be served in one single delivery visit.

Consolidation & Split ($C\&S$): This is our most flexible operational strategy, which allows to split each request into different store tasks that can be served by different shoppers in parallel. As in C, shoppers can simultaneously shop tasks of multiple requests.

4.1 The effect of request splits

Intuitively, more shopper scheduling options are available when requests can be 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 2 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 also a larger set of feasible routing options; we call it the potential *routing benefit* of splitting requests. The example in Figure 2 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, which we call *shopping consolidation benefit*. However, consolidation

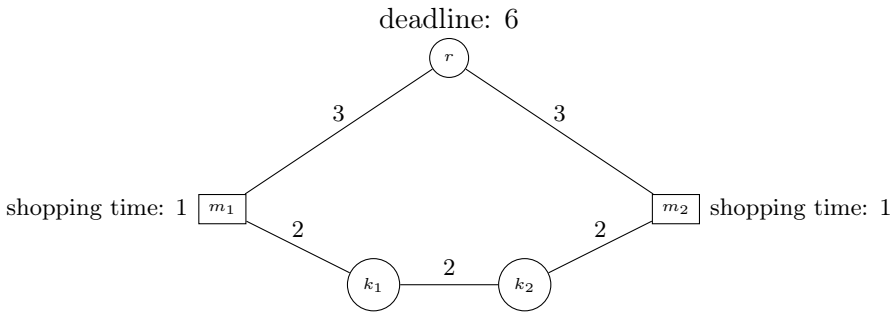


Figure 2: Example of a personal shopper service instance

opportunities are not limited to splitting requests and strategy C can also obtain a fraction of these savings.

4.2 Idling strategy

We assume that each shopper k leaves a location as soon as possible if there exists subsequent visits scheduled in trip τ_k. Otherwise, the platform employs a *nearest store* strategy, *i.e.*, relocates the shopper to the closest store. In preliminary experiments, we also tested different shopper waiting strategies, but did not produced time savings. It seems that waiting strategies in this setting are not required. Probably, because the platform can always update a plan, even if a shopper is en-route.

5 Solution Approaches for the Shopper Routing Problem

In this section, we present a heuristic approach to solve the static *Shopper Routing Problem* (SRP) that we need to solve for all active tasks each time a new request arrives. Moreover, we also describe an exact method that can be used as a benchmark to assess the performance of the heuristic. In particular, we first explain the approach for the most general strategy, i.e., $C\&S$, and then explain how to adapt it to simpler strategies.

The SRP model aims to minimize total working time and can be modelled on a task-based graph $G = (V, A)$ where each active shopping and delivery task s of a request r defines two nodes s^+ and s^- representing its pickup and a delivery activities, respectively. The set of nodes V contains all task-based nodes over active all tasks in S^a and current shopper locations.

An arc in set A links each pair of task-based nodes $i, j \in N$. We define c_{ij} 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 represents a shopping task at a store, then $c_{i,j}$ also includes shopping times. Appendix 8 presents the formal definition of graph G and costs for each arc in set A.

In G, the SRP can be formulated as a Pickup and Delivery Problem with Time Windows. However, we may exploit that path costs are sequence independent if a path of task-based nodes represent shopping operations at a common store location. 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 contain the same node sequences except partial sub-sequences X and $X^{'}$. Also, let X and $X^{'}$ be two different permutations of a given set of nodes in G representing task within the same location. Then, τ and $\tau^{'}$ have the same trip duration.*

5.1 Smart brute force

This section describes an exact approach to solve the SRP which consolidation and split are allowed. In particular, we enumerate all task-to-shopper partitions and then determine minimum duration trips for each shopper. Let $p = \{P_1, \cdots, P_{n_K}\}$ 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. For a given set of shopper tasks P_k, we can identify a trip τ_k that serves all tasks at minimum costs by solving a Hamiltonian path problem with Precedence Constraints and Deadlines. The set of optimal trips for all shoppers in partition p then corresponds to an optimal delivery plan for p.

To identify which partition minimizes the total service time, we explore all partitions sequentially, but prune some unattractive partitions *a priori* via bounding rules similar to a B&B tree search. At any moment, we store the best partition found and its objective value as an upper-bound to the optimal value, *i.e.*, an incumbent solution. For each unexplored partition, we start solving Hamiltonian path problems for each shopper. Given a set of already optimized shopper trips within p, their total time lower bounds the partition's minimum total service time. As soon as this partial cost exceeds the incumbent value, we discard the partition. The approach terminates when all partitions are checked. The final incumbent solution is a 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, v, 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 $(X \cup \{j\}, v, j)$ with the following REFs: $v_j = REF_{ij}(v_i) := v_i + c_{ij}$. An extended (partial)

trip can be infeasible, feasible but dominated, or non-dominated. We eliminate all trips that are dominated.

Algorithm 1 describes the forward labeling algorithm for shopper k over tasks P_k. Let $\mathcal{X}_l$ be collection of partial trips with length $l = |X| - 1$. Subroutine $FeasibilityCheck()$ 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 $BoundingCheck()$ checks 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 1 Forward labeling algorithm

```
Set X_0 = {X = {ω_k}, 0, ω_k}
for l = 1 ··· |P_k| − 2 do
    for x ∈ X_l do
        for j ∈ P_k, j ∉ X do v_j = REF_ij(v_i)
            if FeasibilityCheck(v_j) then
                if BoundingCheck(v_j) then
                    Add x_j to X_{l+1}
                end if
            end if
        end for
    end for
    DominanceRule(X_{l+1}),
end for
Find a path x with label (P_k, v, ·) such that c is minimal.
```

We eliminate partial trips according to the following dominance rule:

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

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.

The exact approach was designed for $C\&S$, but can be easily adapted to simpler operational strategies such as C and $1b1$. These last two only limit the set of feasible routes and partitions and therefore simplify the search.

5.2 Heuristic approach

For larger instances, it is too time-consuming to optimally solve the SRP. Therefore, we propose a heuristic approach that consists of the following steps: (i) Generate multiple initial feasible solutions and (ii) Improve each initial solution by adaptive large neighbourhood search (ALNS). From the set of final solutions, we select the solution that minimizes our objective value.

To generate an initial solution, we use a randomized variant of cheapest insertion. That is, we sequentially insert tasks at the best position of the shopper route plan. At each step, we randomly insert a task from a set of potential candidates that have low insertion costs. More precisely, the candidate set consists of tasks with an insertion cost not exceeding $c_{min} + \lambda \cdot (c_{max} - c_{min})$, where c_{max} and c_{min} are the maximum and minimum insertion costs, respectively.

Each initial solution is later improved by an adaptive large neighborhood search. The neighbourhoods are defined by the following destroy and repair operators.

Destroy Operators. For a number of n_s^a tasks and n_r^a requests in the incumbent solution, we set a removal ratio $\rho \in [0, 1]$ and, depending on the following five operators we remove $q_s = \rho \cdot n_s^a$ tasks or $q_r = \rho \cdot n_r^a$ 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.

Within the ALNS framework, the incumbent solution is destroyed and repaired by the different operators until the stopping criteria is met. The destroy and repair operators are selected according to their weights (ω). The weights are continuously updated based on the success in finding better solutions in previous iterations. After each ALNS iteration, we use local search to further improve the different shopper trips, i.e., by applying *2opt* and *1-point* moves.

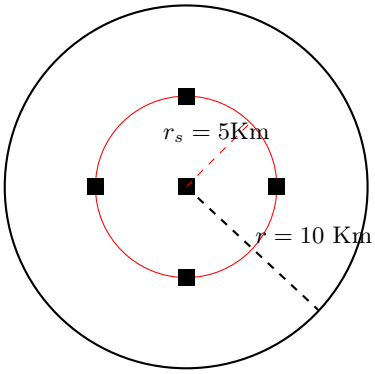
The heuristic can easily be modified to accommodate only consolidations without splits. In this case, we generate an initial solutions by inserting requests into shopper trips instead of tasks. For the ALNS, we subsequently only use the operators that consider requests.

6 Computational Study

In this section, we present 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.

6.1 Experimental setup

Figure 3: Layout of the service region in our experiments



For our experiments, we define a circular service area with a 10km radius and five stores: one in the center and four located at equal intervals of an inner circle with a 5km radius (see Figure 3). The customers are uniformly spread throughout the service region.

The base case experiment considers instances with 80 requests, that arrive over a ten hour service period. We consider exponential request inter-arrival times and, thus, realizations are uniformly distributed within the service period.

A request requires items to be shopped at one or more store locations. In particular, we assume the number of tasks per request to be uniformly distributed between 1 and 4.

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 $f_m = 9$ and $p_s = 1$, meaning that a shopper spends nine minutes for each store visit and one minute per task within a store. Table 1 summarizes all parameter values used in the base case experiment.

6.2 SRP heuristic validation

Now, we evaluate the performance of our heuristic compared to the exact solution of the SRP for the snapshot problem. We execute a rolling horizon framework as described in Section 3.3. To enable us to solve the associated snapshot problems optimally, we use an instance with 40 request and two

Table 1: Base Case Parameters

Number of arriving requests	80
Shopper speed	30 km/h
Shopping time parameters:	
$f_m = f$	9 min
$p_s = p$	1 min
Deadline	90 min
Shopper capacity	10 tasks

shoppers here. We draw five instances at different decision epochs, where the number of active tasks varies between 8 and 15.

Table 2 presents the average percentage difference in total service time between the heuristic and the exact approach, and the number of times our heuristic finds the optimal solution. More importantly, we show the number of times that our heuristic fails to find a feasible solution while the exact approach does find one.

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

n_s	Av. Opt. gap (%)	# Optimal	△ Feasibility
8	1.3	4/5	0
9	1.5	4/5	0
10	4.0	3/5	0
11	0.8	4/5	0
12	2.1	3/5	0
13	2.8	4/5	0
14	3.7	1/5	0
15	3.1*	0/5	1

*Average over four instances

The results show that the heuristic performs well with an average optimality gap of less than 4.0%. Although it does not always finds the optimal solution, the proposed heuristic identifies a feasible solution most of the time and, thus, generally requires the same number of shoppers to serve all tasks as the exact approach.

6.3 Base case results

Table 3 presents the results for the different strategies averaged over 100 random streams of request arrivals. Each realization was solved with a shopper fleet size K set equal to the minimum required to cover all requests for each particular operational strategy.

We report the following performance indicators: (i) *Time per request*: The total service time of all shoppers combined divided by the number of requests. We further break this value down in two

parts: *Shopping time* and *Travel time* per request. (ii) *Travel time per request*: the total travel time divided by the number of requests. (iii) *Number of Shoppers*: the average of the minimum number of shoppers required to serve all requests. (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 earliest and latest delivery of a split request, measured as the sum of all time intervals divided by the total number of split requests.

(vi) *Requests split*: the number of split requests as a percentage of total served requests.

Table 3: Average base case results, **80 Requests, L: 90 minutes**

	DIY	1*b*1	*C*	*C*&*S*
Time per Request (min.)	48.2	42.0	25.8	21.6
Shopping Time per Request (min.)	24.9	24.9	13.5	8.5
Travel Time Per Request (min.)	23.4	17.1	12.4	12.6
No. of Shoppers	NA	7.8	4.9	4.0
Click-to-door (min.)	48.2	49.7	73.5	66.5
Request split (%)	NA	NA	NA	33.3
Delivery Interval (min.)	NA	NA	NA	25.4

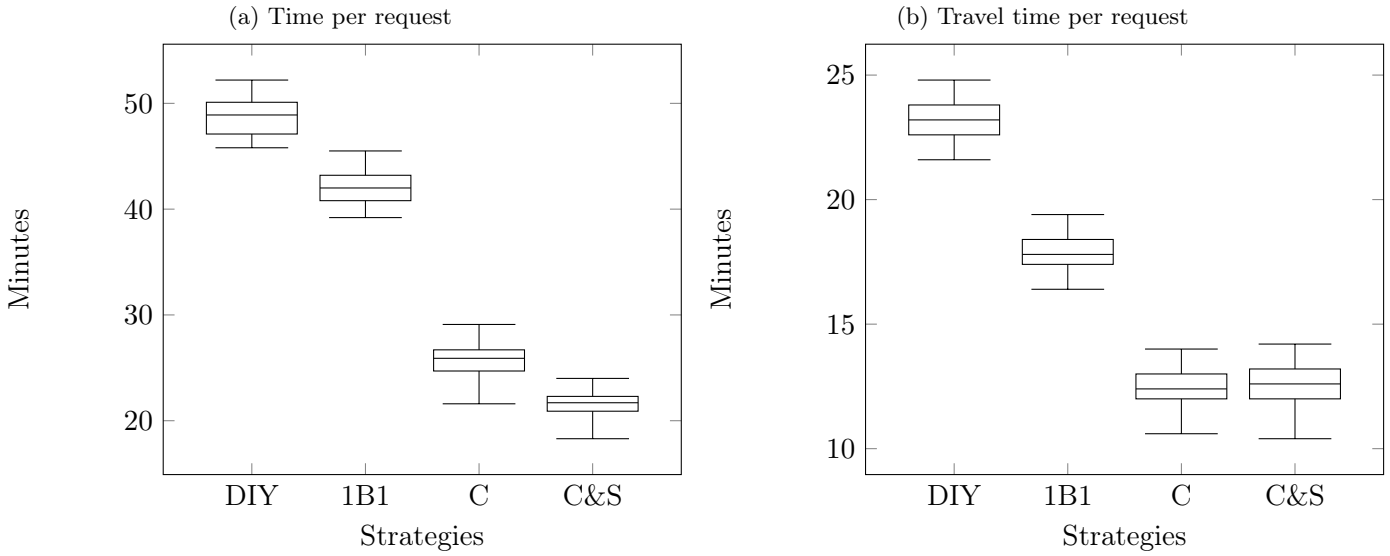
On top of three proposed operational strategies, described in Section 4, we generate a setting in which customers carry out shopping activities by themselves, referred to as the Do-it-Yourself (DIY) benchmark. Here, we assume that each customer chooses the fastest route to visit all stores that are part of the shopping request. Also, we assume that each customer leaves her home at the time when she would otherwise place a request.

The results show that personal shopper services can provide substantial savings as compared to the *DIY* benchmark. Even for the simple 1*b*1 policy, we see savings of 12.9% and 31.3% in the total time spend and time spent for travel respectively. These savings are possible as the personal shoppers can efficiently link the start and end points of consecutive single customer trips. Note that this efficiency gain only leads to marginally higher click-to-door times for 1b1 as compared to *DIY*, i.e, 48.2 versus 49.7 minutes.

Even more savings are possible from consolidating customer trips, *i.e.*, strategy *C*. The time and distance savings further improve with 46.5% and 34.5% as compared to *DIY*. Moreover, consolidation reduces the number of personal shoppers required. As expected, the average click-to-door time increases when consolidating requests. This is because shoppers serve multiple requests in a single trip extending the average time an item travels en route.

An additional performance increase is possible when request splitting is allowed, *i.e.*, *C*&*S*. The time per requests decreases from 25.8 to 21.6 (16.3%). Request splitting enables more consolidation opportunities by reducing task 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 12.4 to 12.6. This could relate

Figure 4: Base Case Results, **80 Requests, L:90 minutes**



to the fact that splitting means that a request destination is visited multiple times. Interestingly, we see that splitting allows for smaller click-to-door times on average probably because multiple tasks belonging to the same request are attended by different shoppers in parallel.

For $C\&S$, we see that on average 33.3% of requests are split and thus served by more than one shopper. For the orders that are split, the average time between the first partial delivery and the final partial delivery, *i.e.*, the delivery interval, is 25.4 minutes.

To provide more insights into the solutions per instance, Figure 5 shows box plots of the time and travel time per requests for the different strategies. We clearly see that the personal shopper service outperforms the DIY benchmark in all instances for all strategies both in terms of total time and travel time per request. Moreover, we observe that the more sophisticated strategies C and $C\&S$, that involve splitting and/or consolidation, outperform the simple 1b1 strategy by a substantial margin.

6.4 Impact of the number of tasks per request

Up to now, we considered instances for which the number of tasks per request varied between the requests. To study the impact of the number of tasks per request in isolation, in this experiment we consider instances in which each request consists of one specific number of tasks. Recall that each task corresponds to shopping at a specific store location. To allow for comparison, the same stream of requests is used in each of the different scenarios, only varying the number of tasks per request.

Our results are summarized in Figure 5. Figure 5a shows the time per request for the different

numbers of tasks per request and strategy. As expected, the time per request increases with the number of tasks. However, we observe that the increase is less steep for strategies C and $C\&S$ as compared to DIY and $1b1$. The reason for this is that these strategies allow shoppers to combine multiple tasks that are associated with a specific store. $C\&S$ has more opportunities to do this than C which leads to additional benefits.

Figure 5b presents the travel time per request for the different numbers of tasks per request. In line with the total time per request, the travel time per request grows with the number of tasks. Here, we see not much difference between C and $C\&S$ which suggests that most benefits of splitting are in terms of shopping time.

Figure 5: Impact of Request Size, **80 Requests,** $\alpha : 0.9$**, L: 90 minutes**

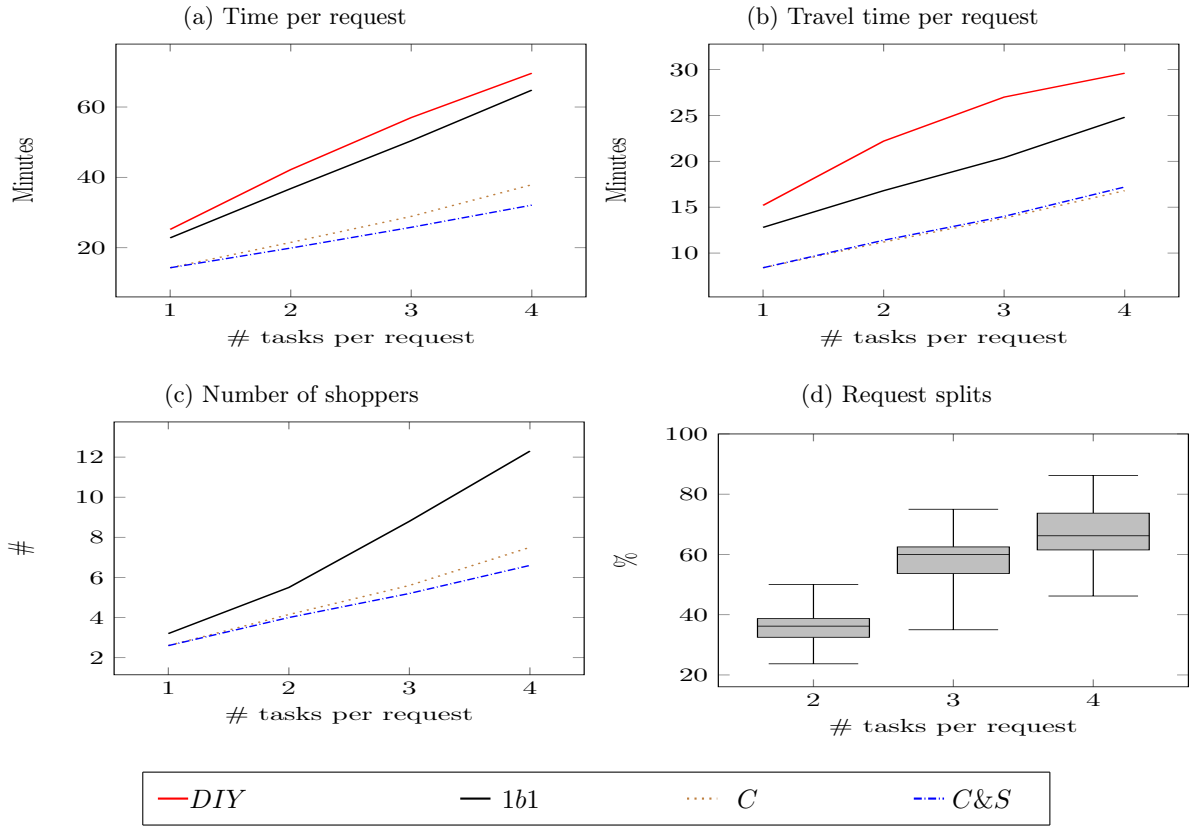


Figure 5c reports the number of shoppers required to serve all requests. As expected, the number of required shoppers increases with the number of tasks per request for all strategies. The reasons for this is that the shopping and travel time increase with more store visits per request. However, we observe that the increase is less pronounced for the strategies with consolidation (C and $C\&S$) as compared to the simple $1b1$. Moreover, also here we see the benefits of splitting.

The box plots in Figure 6d provide insights into the solutions for the $C\&S$ strategy as it shows

that the number of requests with at least one split increases with the number of tasks per request. These results are intuitive as the number of splitting options increases with the number of tasks. We see that the majority of the requests are split in the instances with four tasks.

6.5 Impact of shopping economies of scale α

In this section, we study the impact of the relative weight of the fixed part of the shopping time on the performance of the different proposed strategies. To do so, we set the fixed and variable shopping time parameters equal to $f_m = 10 \cdot \alpha$ and $c_s = 10 \cdot (1 - \alpha)$, respectively, and vary parameter $\alpha \in [0.5, 1]$. A value of $\alpha = 0.5$ indicates that the setup time for a store visit is equivalent to the pickup time per task. This represents the case with little economies of scale. Conversely, a value of $\alpha = 1$ represents a situation with the highest possible economies of scale, in which store shopping times are fixed and independent of the number of tasks shopped at the store. In our experiments, the total store shopping time per visit is independent of α if no consolidation occurs.

Figure 6 presents the performance for the different values of α over all strategies. Figure 6a reports that the average time per request decreases with α. This is intuitive as the value of consolidating, in terms of economies of scale for shopping, increase with α. As expected, α does not effect the performance of the $1b1$ and DIY strategy as it does not allow consolidations.

Figure 6b also shows that for strategies with consolidation (C and $C\&S$), the total travel time per request decreases with α as each shopper visits less stores. Interestingly, we see that with a lower consolidation factor α, $C\&S$ is associated with less travel time than C. This pattern illustrates the flexibility of the $C\&S$ strategy.

Both Figure 6a and 6b present the major benefit of request splits in shopping time. With consolidation (C and $C\&S$), the average shopping time per request decreases as α 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.

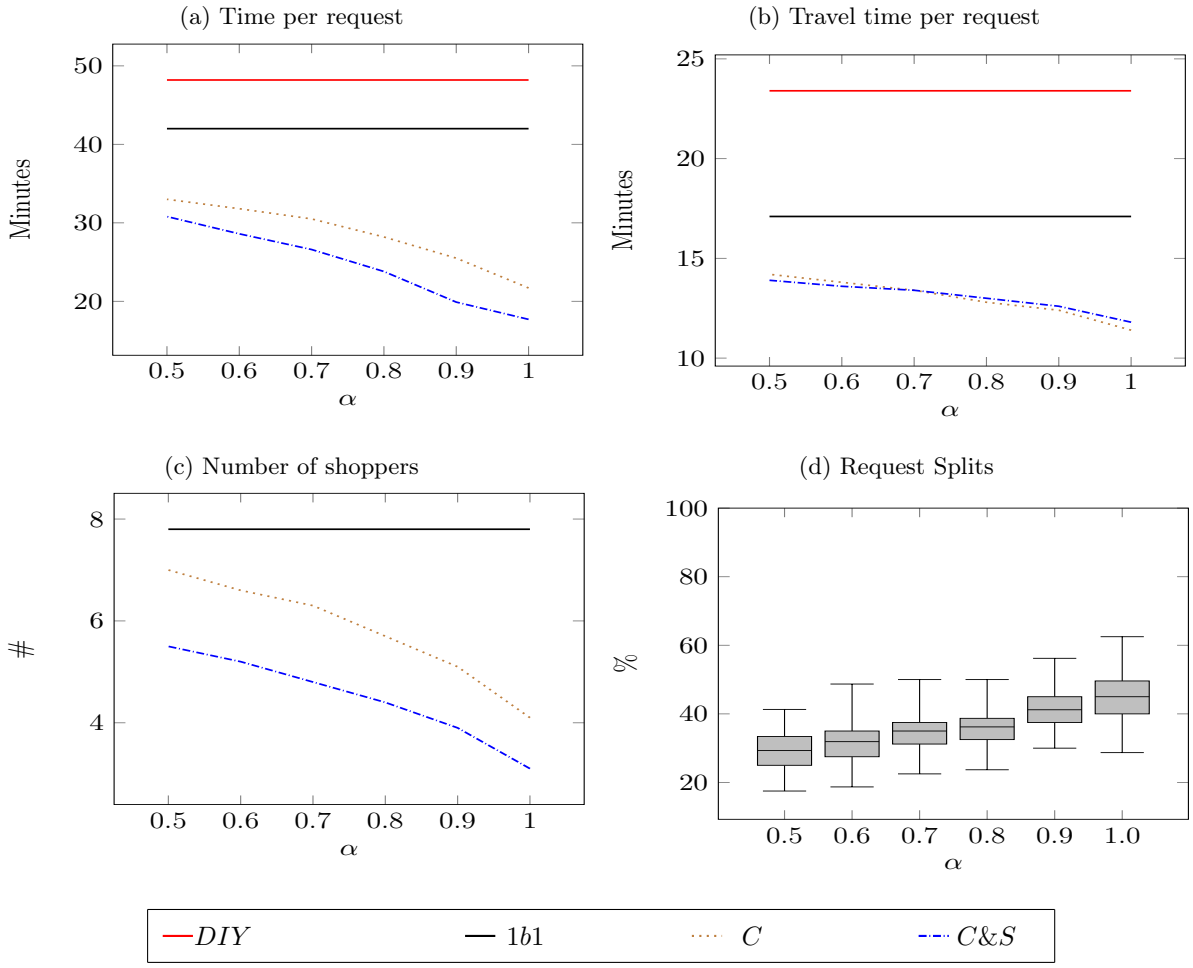
Similarly, we observe in Figure 6c that for strategies C, and $C\&S$ the number of shoppers reduces with α. This is a consequence of the decreasing time per request associated with the consolidation and split strategies.

Figure 6d shows that splitting increases with α. This is intuitive as the potential shopping benefits increase with α which makes it more beneficial to split.

6.6 Impact of number of splits

In practice, not all customers may accept split deliveries. We model the customers' willingness to accepts splits by a parameter $\rho \in [0, 1]$ indicating the probability that a customer agrees to split delivery. For example, $\rho = 0.2$ means that the probability that a customer will accept a split is

Figure 6: Impact of Shopping Consolidation Factor, **80 Requests, L: 90 minutes**



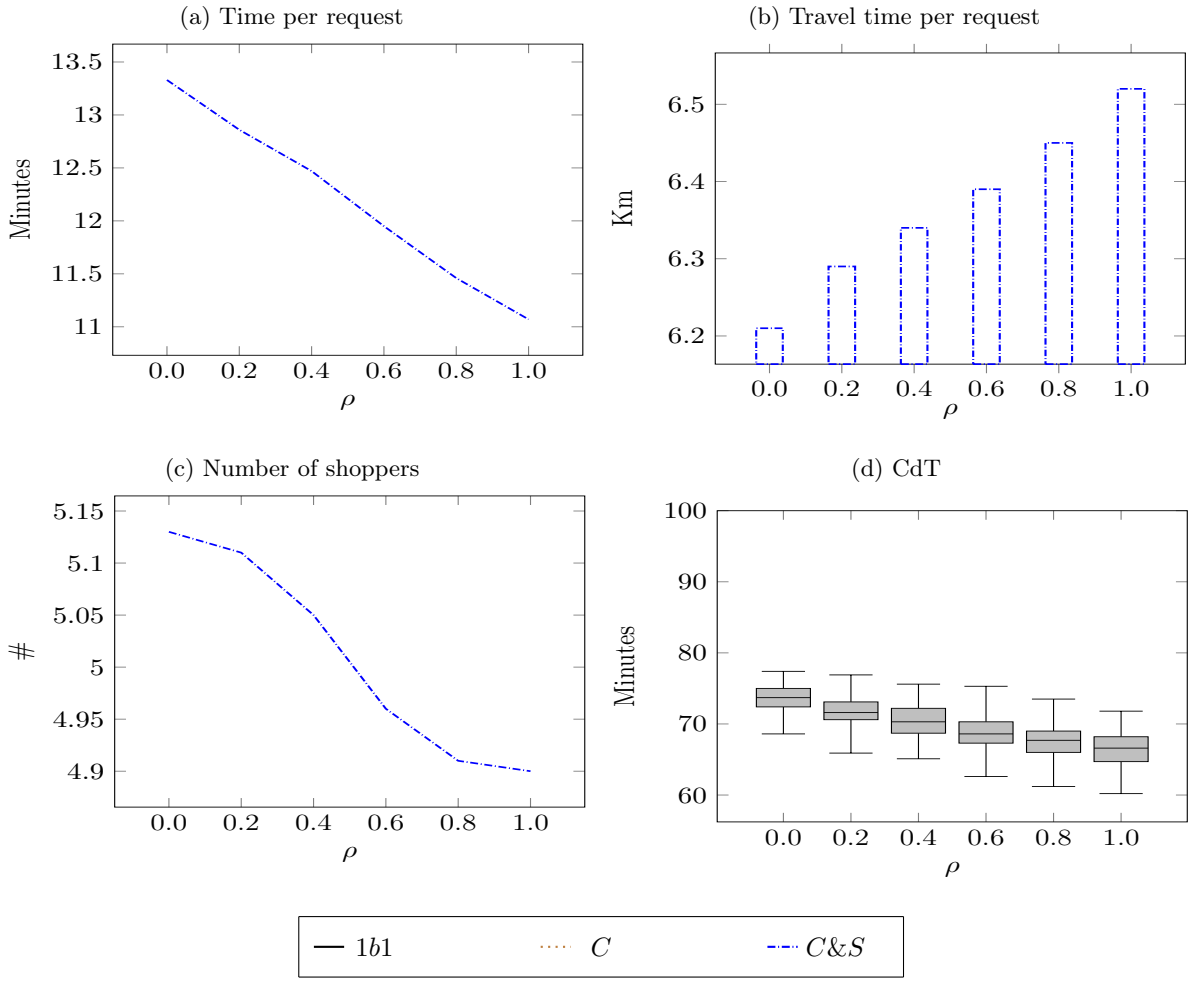
20%.

Table 7 reports the results for different ρ values. As expected, we see that the different performance measures improve with ρ. This is intuitive as a higher ρ corresponds to a less constrained instance. Interestingly, the improvement increase appears to be proportional to the increase in ρ. This suggest that while the savings may vary dependent on the specific set of customers that allow splitting, the average effect is determined by the number of potential splits.

7 Concluding Remarks

The paper focuses on a personal shopper service that provides on-demand delivery from local brick and mortar stores. The service provider uses a fleet of personal shoppers to serve all customer requests while aiming at minimizing total service time. We model the problem as a sequential decision problem and present a rolling horizon approach to solve it. Moreover, we present and

Figure 7: Impact of Split Applicable Request Ratio, **80 Requests, L: 90 minutes**



evaluate three operational strategies for the personal shopper service.

We have conducted an extensive computational study to compare the performance of the different operational strategies and also benchmark against a setting in which customers go to the store to shop themselves. Our results provide the following insights: (i) Personal shoppers can save time and resources over customers shopping by themselves, even when shoppers serve customers one by one. (ii) Shopping and travel times is reduced considerably further by consolidating requests in shopper trips. (iii) It is possible to further enhance the performance of the system by splitting requests that require shopping at multiple stores.

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 plan decisions considering probabilistic information on the placement, location and size of future customer requests. Such information could help us to estimate the expected cost of present decisions in future

states of the PS service operation. It can be valuable knowledge when the volume of future request realizations per unit of time changes significantly throughout the day with a considerable fraction of predictable stationarity. This would be the case of consolidated PS services with well-known demand patterns.

Another interesting research avenue involves incorporating both splits and transfers between personal shoppers see e.g, Macrina et al. (2017, 2020). This means allowing transfers of shopped tasks between shoppers to consolidate delivery operations and enhance customer service while utilizing the part of request split benefits.

References

Archetti, Claudia and Maria Grazia Speranza, "The split delivery vehicle routing problem: a survey," in "The vehicle routing problem: Latest advances and new challenges," Springer, 2008, pp. 103–122.

_ , 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.

Arslan, Alp M, Niels Agatz, Leo Kroon, and Rob Zuidwijk, "Crowdsourced delivery: A dynamic pickup and delivery problem with ad hoc drivers," *Transportation Science*, 2019, *53* (1), 222–235.

Gils, Teun Van, Katrien Ramaekers, An Caris, and René BM de Koster, "Designing efficient order picking systems by combining planning problems: State-of-the-art classification and review," *European Journal of Operational Research*, 2018, *267* (1), 1–15.

Henn, Sebastian, Sören Koch, and Gerhard Wäscher, "Order batching in order picking warehouses: a survey of solution approaches," in "Warehousing in the global supply chain," Springer, 2012, pp. 105–137.

Holsenbeck, Kathie, "Everything you need to know about Prime Now," 2018. Accessed on 12/12/2018.

Jane, Chin-Chia and Yih-Wenn Laih, "A clustering algorithm for item assignment in a synchronized zone order picking system," *European Journal of Operational Research*, 2005, *166* (2), 489–496.

Klapp, Mathias A, Alan L Erera, and Alejandro Toriello, "The dynamic dispatch waves problem for same-day delivery," *European Journal of Operational Research*, 2018, *271* (2), 519–534.

_ , _ , and _ , "The one-dimensional dynamic dispatch waves problem," *Transportation Science*, 2018, *52* (2), 402–415.

_ , _ , and _ , "Request Acceptance in Same-Day Delivery," *Working paper, Georgia Institute of Technology*, 2019.

Koster, René De, Tho Le-Duc, and Kees Jan Roodbergen, "Design and control of warehouse order picking: A literature review," *European journal of operational research*, 2007, *182* (2), 481–501.

Macrina, Giusy, Luigi Di Puglia Pugliese, Francesca Guerriero, and Demetrio Laganà, "The vehicle routing problem with occasional drivers and time windows," in "International Conference on Optimization and Decision Science" Springer 2017, pp. 577–587.

_ , _ , _ , and Gilbert Laporte, "Crowd-shipping with time windows and transshipment nodes," *Computers & Operations Research*, 2020, *113*, 104806.

Nowak, Maciek, Özlem Ergun, and Chelsea C White III, "Pickup and delivery with split loads," *Transportation Science*, 2008, *42* (1), 32–43.

_ , Ozlem Ergun, and Chelsea C White III, "An empirical study on the benefit of split loads with the pickup and delivery problem," *European Journal of Operational Research*, 2009, *198* (3), 734–740.

Pillac, Victor, Michel Gendreau, Christelle Guéret, and Andrés L Medaglia, "A review of dynamic vehicle routing problems," *European Journal of Operational Research*, 2013, *225* (1), 1–11.

Rey, Jason Del, "InstacartâĂŹs new $7 billion valuation is a bet on the future of grocery delivery âĂŤ not a wager against Amazon," 2018.

Reyes, Damian, Alan Erera, Martin Savelsbergh, Sagar Sahasrabudhe, and Ryan ONeil, "The Meal Delivery Routing Problem," *Optimization Online*, 2018.

Sampath, Uday, "Instacart to expand grocery pick-up service to all 50 states by end of 2020," 2020.

Savelsbergh, Martin WP and Marc Sol, "The general pickup and delivery problem," *Transportation science*, 1995, *29* (1), 17–29.

Srour, F Jordan, Niels Agatz, and Johan Oppen, "Strategies for handling temporal uncertainty in pickup and delivery problems with time windows," *Transportation Science*, 2016, *52* (1), 3–19.

Steever, Zachary, Mark Karwan, and Chase Murray, "Dynamic courier routing for a food delivery service," *Computers & Operations Research*, 2019, *107*, 173–188.

Tilk, Christian and Stefan Irnich, "Dynamic programming for the minimum tour duration problem," *Transportation Science*, 2016, *51* (2), 549–565.

Ulmer, Marlin W, Barrett W Thomas, Ann Melissa Campbell, and Nicholas Woyak, "The restaurant meal delivery problem: Dynamic pick-up and delivery with deadlines and random ready times," Technical Report, Technical Report 2017.

_ , Dirk C Mattfeld, and Felix Köster, "Budgeting time for dynamic vehicle routing with stochastic customer requests," *Transportation Science*, 2018, *52* (1), 20–37.

Voccia, Stacy A, Ann Melissa Campbell, and Barrett W Thomas, "The same-day delivery problem for online purchases," *Transportation Science*, 2017, *53* (1), 167–184.

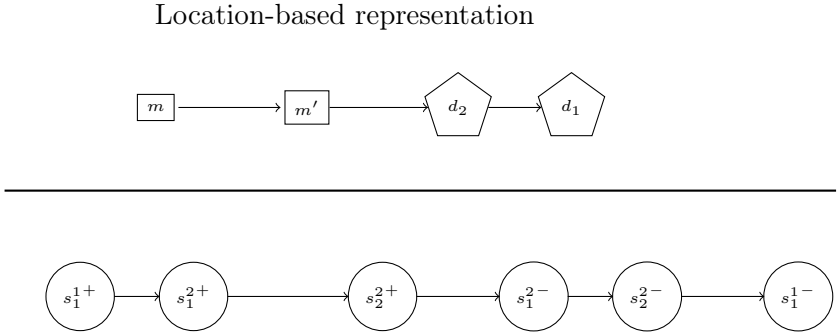
Yildiz, Baris and Martin Savelsbergh, "Provably high-quality solutions for the meal delivery routing problem," *Georgia Institute of Technology*, 2017.

Yu, Mengfei and René BM De Koster, "The impact of order batching and picking area zoning on order picking system performance," *European Journal of Operational Research*, 2009, *198* (2), 480–490.

8 Appendix- Task-Based Graph Representation

Figure 8 illustrates the how we can derive a task-based graph from a location-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_2^{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.

Figure 8: Location-based and task-based graphs for a particular shopper's route. Pentagons and squares indicate customer and store locations, respectively. Circles represent shopping and delivery nodes.



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

In equation 2, 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 $i \overset{\text{L}}{\neq} j$ that checks whether i and j are two different physical locations.

$$c_{ij} := \mathbb{1}_{\left(i \overset{\text{L}}{\neq} j\right)} t_{ij} + \mathbb{1}_{(j \in S^+)} \left(c_j^p + \mathbb{1}_{\left(i \overset{\text{L}}{\neq} j\right)} c_j^f \right), \tag{2}$$

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