Optimizing Crowdsourced Delivery Routes Through Concurrent Selection of Pickup Stores and Drivers

Oscar Correa
University of Melbourne
Melbourne, Australia
oscarcorreag@gmail.com

Egemen Tanin
University of Melbourne
Melbourne, Australia
etanin@unimelb.edu.au

Kotagiri Ramamohanarao
Australian Academy of Science
Melbourne, Australia
rkotagiri@gmail.com

Lars Kulik
University of Melbourne
Melbourne, Australia
lkulik@unimelb.edu.au

Arkady Zaslavsky
Deakin University
Melbourne, Australia
arkady.zaslavsky@deakin.edu.au

Hairuo Xie
University of Melbourne
Melbourne, Australia
xieh@unimelb.edu.au

ABSTRACT

Many retailers are offering crowdsourced delivery where ad hoc drivers collect goods from pickup stores and deliver the goods to customers on behalf of the retailers. Efficient spatial data management solutions are needed to optimize the routes of the drivers. In the existing model of crowdsourced delivery, retailers make delivery plans regardless of drivers' original full routes. This can lead to unnecessarily high delivery costs. We propose a novel crowdsourced delivery model that optimizes delivery routes by concurrently selecting pickup stores and drivers. Based on this model, we develop a heuristic solution for crowdsourced delivery. Experimental results show that our solution saves delivery costs by up to 50% compared with the existing model used by retailers. Our solution is also scalable for large cities.

CCS CONCEPTS

• **Applied computing** → *Transportation.*

KEYWORDS

spatial data management, route optimization, heuristic algorithms

ACM Reference Format:
Oscar Correa, Egemen Tanin, Kotagiri Ramamohanarao, Lars Kulik, Arkady Zaslavsky, and Hairuo Xie. 2022. Optimizing Crowdsourced Delivery Routes Through Concurrent Selection of Pickup Stores and Drivers. In *The 15th ACM SIGSPATIAL International Workshop on Computational Transportation Science (IWCTS '22), November 1, 2022, Seattle, WA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3557991.3567781

1 INTRODUCTION

Many retailers are offering crowdsourced delivery service, where ad hoc drivers earn a fee for picking up goods and delivering the goods to customers. For example, Walmart offers a crowdsourced delivery service called Spark Delivery in the US. The choice of the delivery routes has a direct impact on delivery costs and delivery speed. Efficient spatial data management solutions are needed for optimizing the routes in citywide road networks based on a large amount of spatial data such as the location of stores and customers. When planning the delivery routes, the existing crowdsourced delivery services are based on a model where a retailer first chooses the pickup store for a request and then assigns the delivery job to the driver who is the closest one to the chosen store at that time. As the existing model does not consider the original route of drivers when making delivery plans, the delivery routes can be suboptimal with high travel costs. The difference between a suboptimal delivery route and an optimal delivery route can be seen in Figure 1, where the delivery cost is reduced when the selection of pickup store is optimized based on the original route of an ad hoc driver. This is a simple example out of many suboptimal cases.

We propose a crowdsourced delivery model that minimizes delivery costs through concurrent selection of pickup stores and drivers when planning the delivery routes. Our model is fundamentally different from the existing model adopted by retailers in that our model aims to reduce the detour of ad hoc drivers at the global level by exploiting the original route of ad hoc drivers, the location of all pickup stores and the location of customers. We call this model

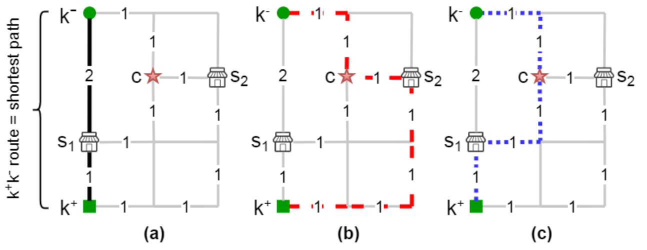

Figure 1: (a) Customer c placed a request to a retailer that owns pickup stores s_1 and s_2. The delivery request is assigned to an ad hoc driver who is going from work to home and whose original route is the shortest path between k^+ and k^-. (b) Without considering the driver's original route, the retailer may choose s_2 as the pickup store at order placement time, resulting in a suboptimal delivery route with cost 7 (red dashed line). (c) By considering the driver's original route, the retailer would choose s_1 as the pickup store, resulting in a better delivery route with cost 5 (blue dotted line).

Crowdsourced Delivery through Concurrent Routing and pickup Store Selection (CD-CRSS). We develop an efficient spatial data management solution to solve instances of CD-CRSS in large cities, where a large number of delivery requests need to be processed at the same time. We believe our direction will reduce delivery costs for retailers and increase crowdsourcing as well.

Our solution involves two stages, an assignment stage and a routing stage. In the assignment stage, our solution assigns delivery jobs to drivers such that the estimated travel cost of all the drivers is minimized. This stage will be presented as a basic nearest neighbor-based algorithm and then as a more advanced algorithm. The later algorithm starts with a naive assignment plan then gradually optimizes the plan with a minimal spanning tree using a heuristic. This stage solves a new type of spatial query: Given a set of drivers and the predecessors of a customer (the candidate pickup stores), the query returns the driver whose delivery route, via one of the predecessors, to the customer, has the lowest travel cost. This query is a variant of the In-Route Nearest Neighbor (IRNN) query [32, 35]. We name our new query *In-Route Nearest Neighbor with Predecessors*.

In the routing stage, our solution computes the **exact** delivery routes for drivers who are assigned customers. This stage will be presented as a basic nearest neighbor-based routing algorithm and then as a more advanced routing algorithm that uses a branch-and-bound approach. The routing stage solves a new routing problem, which aims to find the shortest route that passes a given set of nodes (the customers) via the predecessors (the candidate pickup stores). This new routing problem is a variant of the shortest Hamiltonian path problem [31]. We name the new routing problem *Group Hamiltonian Path with Precedence Constraints*.

To further manage delivery costs, we develop two novel detour control methods. The methods can help limit the distance of detours for reaching pickup stores or customers. The first method guarantees that a driver only visits stores and customers that are within a certain distance to his or her original route. The second method sets a stricter limitation on travel cost such that the ratio of the distance of the delivery route to the distance of the original route must be below a threshold no matter the number of stops.

To the best of our knowledge, none of the existing solutions for crowdsourced delivery can minimize travel costs in such a comprehensive way as our solution. Our CD-CRSS model can be applied to an application scenario where an ad hoc driver picks up goods from one retailer and delivers the goods to multiple customers of the same retailer. The model can also be applied to a more generic case where the driver delivers goods to multiple customers who buy from different retailers. In this paper, we focus on the generic case. Our solution is guaranteed to achieve the same or lower delivery costs than the state-of-the-art solutions because our solution considers more factors when optimizing delivery routes. For example, our solution considers all the available pickup stores of a retailer for a customer while the state-of-the-art considers only one.

Our experiments show that a cost saving of 50% can be achieved when CD-CRSS is applied to a retailer-independent platform like Amazon, where customers can choose from a wider range of retailers. Our solution is readily deployable as it runs as fast as the existing model used by Walmart and alike. Our solution can even serve large cities efficiently. Also, our solution runs up to 3 orders of magnitude faster than a linear programming-based solution.

2 RELATED WORK

2.1 Crowdsourced Delivery

Many existing works in crowdsourced delivery [2–4, 6, 7, 9–12, 14, 15, 17, 19, 25, 26, 30, 34, 36] **assume that the purchased items of one customer can only be picked up from one specific location. This is different to our work as we note the fact that in many cases there can be multiple pickup locations for one customer from a retailer**. The only work that has studied a similar configuration is [37]. None of the other existing solutions considers the optimization of pickup location selection when planning driver routes. From this point of view, our model (CD-CRSS) is in a highly advantaged position as it optimizes from multiple perspectives at the same time. The existing works mainly fall into two categories depending on whether deliveries are fully crowdsourced or not. In the first category, deliveries are fully crowdsourced, which means the ad hoc carriers are responsible for picking up items and delivering the items to customers directly [4, 6, 7, 9, 10, 12, 19, 26]. In the second category, deliveries are partially crowdsourced, where ad hoc carriers only move parcels between intermediate depots in road networks [15, 30, 34]. The customers need to drop off the parcels or pick up the parcels at the depots by themselves. Our work falls into the first category as we assume that ad hoc drivers need to visit stores and then customers directly.

A recent work [33] on shared mobility applications proposes a route planning method that can be applied to ride-sharing and food delivery, which can be adapted to crowdsourced delivery as well. This work also assumes that there is one pickup location for one customer, that is, it does not consider the selection of pickup locations for individual customers. Due to this reason, the approach cannot be applied to the application scenarios addressed in our work. We also note that traditional ride-sharing approaches [1, 5, 16, 21, 23, 24] inherently assume that there is only one pickup location. Therefore, they cannot be used to solve CD-CRSS either.

Another recent work, Online Trichromatic Pickup and Delivery Scheduling (OTPD) [37], develops a crowdsourced delivery solution where a set of pickup points are available for each request. OTPD selects pickup locations and drivers based on various constraints, such as the expiration time of delivery tasks and the workload capacity of drivers. OTPD is aimed at cases such as food deliveries and its benefits can be observed when real-time deliveries are paramount. Thus, it misses the further optimization that can be achieved by CD-CRSS as our approach exploits the same-day-delivery level of service adopted by big retailers and therefore finds solutions inside batches of requests. CD-CRSS is not meant to solve the online version of this problem. Also, OTPD maximizes a trichromatic utility function whereas CD-CRSS minimizes total travel cost. OTPD is basically aimed to address a stream of queries for perishable goods and thus, an immediate solution is paramount, while we aim to address the common online retail market for next-day delivery or same-day delivery, which gives us room to optimize and address an Amazon-like market. Based on these premises, our approach cannot be benchmarked against OTPD.

2.2 Nearest Neighbor Queries

In the general form, Nearest Neighbor (NN) queries search the closest node to a query point given a set of candidate nodes. Many variants of Nearest Neighbor queries have been studied [18, 29]. There are two types of NN queries that are highly related to our work. One of them is In-Route Nearest Neighbor (IRNN) [32, 35] and another is k-Path Nearest Neighbor (k-PNN) [8]. The two queries are similar as both search the nodes near a given path with the aim to minimize the travel cost via the nodes. The difference between the two queries is that in IRNN the path is fixed whereas in k-PNN the path can change as the starting point of the path is the user's location at real time. Our solution assigns customers to drivers by solving a variant of IRNN query. Our query differs from IRNN as it searches the driver whose delivery route is the shortest one among all the possible delivery routes under the assumption that the route passes one of the pickup stores and then a given customer.

2.3 Hamiltonian Path Problems

Given a set of vertices, a Hamiltonian path is a path that visits each vertex once. The shortest Hamiltonian path problem searches the Hamiltonian path with the lowest cost [31]. The routing algorithm in our solution solves a new type of routing problem, *Group Hamiltonian Path with Precedence Constraints*. The problem is a variant of the shortest Hamiltonian path problem. The new problem requires that certain nodes (customers) can only appear after precedence nodes (pickup stores) on the shortest Hamiltonian path. Our new routing problem is also a generalization of the Sequential Ordering Problem (SOP) introduced by Escudero [13]. Escudero defines the precedence relationships as a directed acyclic graph $P := \langle V, R \rangle$ where an arc $(i, j) \in R$ means that vertex i should precede j, represented as $i \prec j$, in a feasible path. SOP allows the concurrent existence of multiple precedence relationships regarding the same vertex. For example, it allows $i \prec j \wedge k \prec j$, where i and k are two stores of Walmart and j is a customer. However, SOP cannot enforce the selection of precedence relationships as our model does. Taking the previous example, our model can enforce $i \prec j \vee k \prec j$, that is, a Walmart store must be visited before a customer but visiting any Walmart store is sufficient. This cannot be achieved with SOP.

3 PROBLEM DEFINITION

Our research is focused on crowdsourced delivery in large cities, where a large number of online customers may submit delivery requests to retailers within a time interval. These customers can be assigned to ad hoc drivers.

Let $G := \langle V, A \rangle$ be a road network graph with a set of vertices V and a set of edges A. Let **R** be the set of retailers. A retailer $r \in R$ owns a set of stores S_r. The locations of stores and customers are in V for the sake of simplicity without loss of generality. Each edge $e \in A$ is associated with a travel cost, which can be measured as the distance of the edge or some other metric. Let **C** be the set of customers. A request made by customer $c \in C$ contains a retailer name, a list of purchased items, and a delivery address. Let **D** be the set of ad hoc drivers. An ad hoc driver $k \in D$ moves in the road network. The driver follows a path P_k, which is a route that starts at $k^+ \in V$ and ends in $k^- \in V$. The travel cost of the driver is denoted as $\boldsymbol{TC_k}$, which is the sum of the travel costs associated with all the edges on the driver's route. Let **SC** be the service cost of all drivers. SC is measured by travel costs, that is, $SC = \sum_{k \in D} TC_k$. The proportion of customers served by crowdsourced carriers in all the customers is denoted as **PC**.

Our research problem is defined as follows.

Given: a road network G, a set of retailers R, a set of customers C, and a set of drivers D.

Find: a set of routes P such that the **service cost of ad hoc drivers (SC)** is minimized while the **proportion of customers served by the ad hoc drivers in all the customers (PC)** is maximized.

4 OUR APPROACH

To plan for crowdsourced delivery, we adopt a two-stage approach. In the first stage, we assign customers to ad hoc drivers. In the second stage, we compute the **exact** routes of drivers. Computing the optimal plans is computationally difficult because finding the optimal assignment is equivalent to solving the set cover problem, which is NP-hard. For planning crowdsourced delivery in citywide applications, one needs a solution that yields near-optimal plans with low computational costs. Due to this reason, our solution computes the assignment based on heuristics, and only the routing is computed exactly. Our experimental results show that our solution can run several magnitudes faster than a linear programming-based solution that computes the optimal delivery plan. Meanwhile, our solution achieves significant cost savings compared to the existing model adopted by retailers.

4.1 Assignment

Assigning customers to ad hoc drivers is the first stage of our approach. Ideally, customers can be assigned to drivers while minimizing the service cost (**SC** as defined in Section 3). Computing the optimal assignment is equivalent to solving the set cover problem, which is NP-hard [20, 22]. A set cover problem aims to find the smallest sub-collection of sets such that the sub-collection contains all the elements. Specifically, assume there is a ground set X of n elements and a collection $\mathcal{F}$ of m subsets of X. The goal is to find the minimum number of subsets $S_1, S_2, \ldots, S_h$ in $\mathcal{F}$ such that $\bigcup_{i=1}^{h} S_i = X$, where all subsets in the solution are pairwise disjoint. Note that h can be smaller than m. If each subset in $\mathcal{F}$ has a weight, one can define a Weighted Mutually Exclusive Set Cover Problem, where the goal is to minimize the sum of the weights in the solution. CD-CRSS assignment can be formulated with this problem. Specifically, let C be the set of customers and D be the set of drivers. The ground set X is $C \cup D$. The collection of assignments $\mathcal{F}$ include all the possible assignments, each of which is a subset of X.

In the CD-CRSS assignment stage, we aim to find a solution $\mathcal{F}^* \subseteq \mathcal{F}$, where each subset S in $\mathcal{F}^*$ is associated with a delivery cost $TC(S)$, the cost to serve all the customers in S in one trip. The total $TC(S)$ from all subsets, $\sum_{S \in \mathcal{F}^*} TC(S)$, needs to be minimized. As finding the exact optimal assignments is NP-hard, our solution finds approximate assignments with low computational costs.

We develop two heuristic assignment methods. The first method assigns customers to ad hoc drivers based on the distance between the customers and the original routes of drivers. This can help reduce SC but it does not consider the cost of detours that are needed to reach the pickup stores. To reduce SC further, we develop the second method that not only considers the location of customers and the drivers' original routes but also considers the estimated

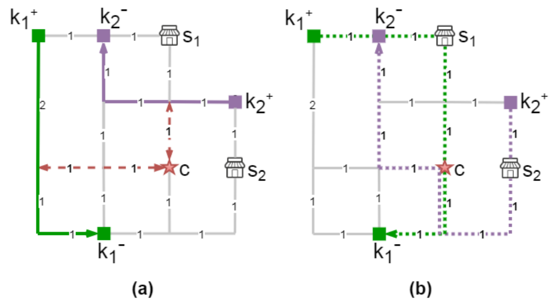

Figure 2: Customer c placed a request to a retailer whose stores are s_1 and s_2. Driver k_1's original route starts from k_1^+ and ends in k_1^-. Driver k_2's original route starts from k_2^+ and ends in k_2^-. (a) Bold lines depict the ad hoc drivers' original routes. Dashed lines show the shortest distances from c to the routes. The nearest neighbor-based method assigns c to k_2 since the distance from c to this driver's route is 1, which is smaller than the distance from c to k_1's route. (b) Dotted lines depict the ad hoc drivers' potential delivery routes. With the *In-Route Nearest Neighbor with Predecessors query*, c is assigned to k_1 since k_1's potential route via s_1 (a predecessor of c) has cost 6 while k_2's potential route via s_2 has cost 7.

delivery cost via one of the pickup stores for each customer. We detail both assignment methods in this section.

4.1.1 Nearest-Neighbor-Based Assignment. In this method, we find the suitable ad hoc driver for a customer by solving a variant of the classical Nearest Neighbor query [29]. Given the location of a queried customer and the original routes of individual drivers, the query returns the driver whose route is the closest to the customer. An example of assignment with this method is shown in Figure 2(a).

4.1.2 In-Route-Nearest-Neighbor-Based Assignment. The Nearest-Neighbor-Based assignment is intuitive, yet it overlooks the stores, which can result in suboptimal assignments. For example, a customer may be assigned to a driver whose original route is close to the customer whereas all the pickup stores are far from the route. Driver assignments that consider the estimated cost of delivery via pickup stores would be more effective in reducing SC. A request for such an assignment equates to a novel spatial query, *In-Route Nearest Neighbor with Predecessors*, which asks for the ad hoc driver whose delivery route, via one of the predecessors (pickup stores), to the queried customer, is the shortest one among all possible drivers. An example of this assignment is shown in Figure 2(b).

This assignment method **estimates**[1] the delivery cost of ad hoc drivers based on a distance function, $dist_{IR}$ (Equation 1). In the function, q is a customer, p_k is the original route of driver k with a source k^+ and a destination k^-, S_q is a set of pickup stores (predecessors) for customer q, s is one of the candidate pickup stores, $dist(i, j)$ is the shortest network distance between i and j.

$$dist_{IR}(q, p_k) := \min_{s \in S_q} \{dist(k^+, s) + dist(s, q)\} + dist(q, k^-) \quad (1)$$

[1]Only considers one customer at a time. Recall that an ad hoc driver may serve more than one customer in one go.

Given a customer and all the ad hoc drivers, we can find the most suitable driver for the customer based on the distance function (Equation 1). Driver assignment for all the customers can be found in this way. However, due to randomness in the spatial distribution of customers and pickup stores, this approach may result in unbalanced assignments, where some drivers get overloaded with a large number of orders while others get no orders. Unbalanced assignments may also discourage people to participate in crowdsourced delivery. In addition, overloading some drivers may significantly increase the time on computing the drivers' routes, which affects the scalability of the solution.

To balance the workload between drivers while keeping the estimated delivery cost to the minimum, we develop a load balancing algorithm based on a Minimal Spanning Tree (MST). A MST is a tree where all the vertices are connected while the total weight of its edges is minimized. Our algorithm builds a MST with two groups of nodes, one is customer nodes and the other is driver nodes. The MST maintains the assignment information with the edges between customer nodes and driver nodes. Initially, each customer is connected to a driver that is found based on Equation 1 via an edge. The weight of the edge is the shortest distance computed with the equation. To make all the nodes connected, there is also an edge between any pair of drivers. An inter-driver edge has zero weight so it does not affect the total weight of the MST. By constructing the tree in this way, the total weight of the tree is the total travel cost of the drivers who are assigned customers. In the initial MST, some of the driver nodes may have a high degree, i.e., the corresponding drivers may be overloaded. Our load balancing algorithm aims to minimize the highest degree down to a threshold, **maximum degree m of MST**.

To reduce the degree of the MST, we adopt a greedy technique that iteratively disconnects a customer node from a high-degree driver node and reconnects this customer node to a low-degree driver node, i.e., reassignment of a customer between two drivers. The reassignment process starts by locating the highest-degree driver node in the MST. Among the edges that are connected to this node, we remove the one with the highest weight. This leaves a customer orphan, i.e., without assignment. Then from the low-degree driver nodes, we find the driver with the least travel cost to serve the customer. Finally we connect the orphan customer node to the newly found driver node in the MST. This is called a "local move". The pseudo code of this algorithm is shown below.

```
1:  function MinDeg($G, T, m$)                    ▷ $G$: graph, $T$: MST, $m$: bound
2:      $\mathcal{S} \leftarrow \emptyset$                          ▷ $\mathcal{S}$: drivers on pause
3:      $mov \leftarrow 0$                       ▷ $mov$: local moves between drivers
4:      while True do
5:          $\mathcal{D} \leftarrow$ Degrees($T$)        ▷ $\mathcal{D}$: degrees of not paused drivers
6:          $\Delta \leftarrow \max\{\delta \mid \langle k, \delta \rangle \in \mathcal{D}\}$      ▷ $\Delta$: highest deg. of drivers
7:          if $\Delta = m$ then
8:              break                        ▷ the bound of the degree is reached
9:          if $\Delta = 0$ or ($\Delta = 1$ and $mov = 0$) then
10:             break                        ▷ no local moves were possible
11:         if $\Delta = 1$ then
12:             $\mathcal{S} \leftarrow \emptyset$            ▷ paused drivers are freed to try again
13:             $mov \leftarrow 0$
14:             continue
15:         $v \leftarrow k \in \{k \mid \langle k, \delta \rangle \in \mathcal{D} \wedge \delta = \Delta\}$     ▷ $v$: driver node with
                                                   the highest degree $\Delta$
```

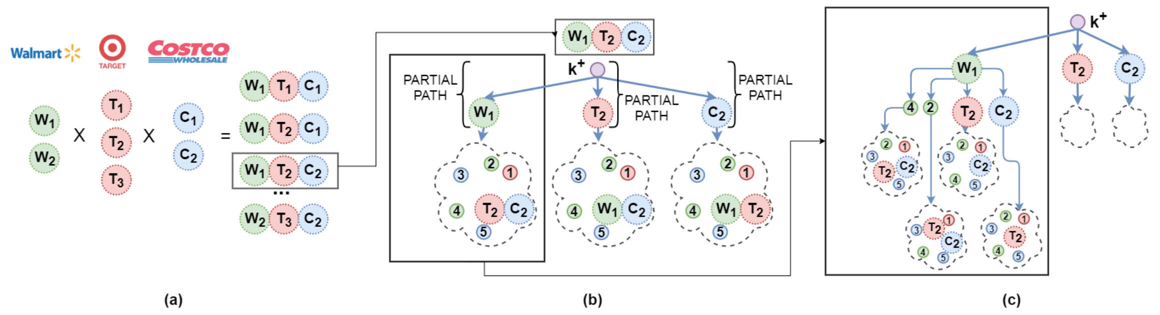

Figure 3: (a) Combinations of stores of three different retailers. Walmart stores (W_1 and W_2) are green. Target stores (T_1, T_2 and T_3) are red. Costco stores (C_1 and C_2) are blue. (b) Three initial partial paths in a BnB tree for a specific combination of stores. Each path starts at the driver's source k^+. A path k^+-W_1 is expanded by branching the tree at W_1. The IDs of customer 1 to 5 are shown in small circles. Note that a retailer and its customers are in the same color. For example, the IDs of two Walmart customers (2 and 4) are shown in two green circles. (c) The path k^+-W_1 is branched into 4 new partial paths, k^+-W_1-*customer* 4, k^+-W_1-*customer* 2, k^+-W_1-T_2 and k^+-W_1-C_2.

```
16:     $\gamma \leftarrow \infty$                          ▷ $\gamma$: upper bound of edge cost
17:     while True do
18:         $Q \leftarrow \{\langle q, c_{v,q}\rangle \mid (v,q) \in E(T) \land c_{v,q} < \gamma\}$
19:         if $Q = \emptyset$ then          ▷ no edges with smaller cost than $\gamma$
20:             $\mathcal{S} \leftarrow \mathcal{S} \cup \{v\}$                  ▷ put $v$ on pause
21:             break
22:         $c_{v,\phi} \leftarrow \max\{c_{v,q} \mid \langle q, c_{v,q}\rangle \in Q\}$
23:
            $\mathcal{K} \leftarrow \{k \mid (k,\phi) \in E(G) \land (k,\phi) \notin E(T) \land$
                $k \notin \mathcal{S} \land \exists \langle k, \delta\rangle \in \mathcal{D} : \delta \leq \Delta - 2\}$

24:         if $\mathcal{K} \neq \emptyset$ then
25:             $c_{v',\phi} \leftarrow \min\{c_{k,\phi} \mid k \in \mathcal{K} \land (k,\phi) \in E(G)\}$
26:             $E(T) \leftarrow E(T) \setminus \{(v,\phi)\}$          ▷ disconnect $\phi$ from $v$
27:             $E(T) \leftarrow E(T) \cup \{(v',\phi)\}$              ▷ connect $\phi$ to $v'$
28:             $mov \leftarrow mov + 1$
29:             break
30:         else
31:             $\gamma \leftarrow c_{v,\phi}$        ▷ by updating the upper bound, $Q$ will
                                  become smaller (line 18) such that the
                                  function will try $v$'s customer associ-
                                  ated with the next largest edge weight
                                  next time (line 22)
32:     return $T$
```

As shown in the pseudo code, from the edges connected to a driver v who has the highest-degree Δ among all driver nodes, we pick the largest-weight edge (v, ϕ) between the driver and a customer ϕ (line 22). If there exists a set $\mathcal{K}$ of drivers who can serve ϕ and have a graph degree of at most $\Delta - 2$, we "move" ϕ to the driver $v' \in \mathcal{K}$ whose $dist_{IR}$ (Equation 1) to ϕ is the shortest among $\mathcal{K}$. If $\mathcal{K}$ is empty, we do not move ϕ and we try to move the customer associated with the next largest-weight edge connected to v (inner `while` loop, lines 17-31). Then, we repeat the whole process with the new highest-degree driver (outer `while` loop). The function finishes when the bound is reached, i.e., $\Delta = m$, or no local moves were possible. Note that Δ can reach values less than m when the driver with the highest degree is put on pause. In this case, the degrees of the drivers with lower degree are minimized instead, and Δ skips the value m. The paused drivers (including the highest-degree driver) are probed again when $\Delta = 1$. The final assignment is given by the resulting MST.

Our experiments show that this method leads to significantly lower service costs (SC) than the basic nearest neighbor-based one.

4.2 Routing

Computing the exact route of ad hoc drivers is the second stage of our solution. Given a road network $G := \langle V, A\rangle$ with a set of vertices V and a set of edges A, an ad hoc driver k, a group of customers assigned to the driver, and all the available pickup stores of the retailers that the customers purchased from, we aim to minimize service cost (SC) by optimizing the delivery route of the driver. We name this novel routing problem as Group Hamiltonian Path with Precedence Constraints Problem (GHPPCP). The classical Hamiltonian path problem aims to find whether there exists a path that visits all the given vertices only once. Finding a delivery route for an ad hoc driver is similar to computing a shortest Hamiltonian path. But GHPPCP imposes further constraints to the shortest Hamiltonian path. First, a pickup store must *precede* its corresponding customers in the route. Second, the route only needs to visit one store per retailer. This is because a driver can pick up the goods for all the customers of a retailer from a single store. In this section, we first present a simple *nearest-neighbor-based* routing algorithm. Then we present a more advanced routing algorithm that uses a branch-and-bound approach.

4.2.1 *Nearest-Neighbor-Based Routing.*

Starting from the source of a driver, this algorithm expands the partial route by appending a pickup store or a customer, which is the nearest stopping point to the last stopping point on the partial route. If the nearest stopping point has been visited or does not need to be visited due to the constraints of the GHPPCP problem, the algorithm keeps searching for the next available nearest stopping point. This is done until the destination is reached.

4.2.2 *Branch-and-Bound Routing.*

This algorithm aims to reduce SC further using a variant of the classical Branch-and-Bound (BnB) approach. A classical BnB algorithm finds the solution to an optimization problem by branching a tree structure. Each branch is associated with a lower bound and an upper bound. The algorithm expands the most promising branch while discarding the branches that would not produce better solutions than the existing best solution. In our case, the root of a BnB tree represents a driver's source. The path from the root to a leaf node represents a partial path between the driver's source and another stopping point, which can be a pickup store, a customer or the driver's destination.

As a driver must visit a pickup store of a retailer before visiting any customer of the retailer, the selection of the pickup store plays a vital role in computing the delivery route. The algorithm enumerates the combinations of pickup stores from different retailers and builds a BnB tree for each combination. An example is shown in Figure 3(a), where the customers purchased from three retailers, Walmart, Target and Costco. The numbers of stores owned by the retailers are 2, 3 and 2, respectively. Therefore, there would be $2 \times 3 \times 2 = 12$ combinations of pickup stores. The example highlights one of the combinations, (W_1, T_2, C_2). A BnB tree is constructed based on this combination. The three stores are in the first level of the BnB tree (Figure 3(b)) as one of them must be the first stopping point after the driver's source. There are 11 other BnB trees constructed in the same way as this tree. A BnB tree may contain a number of partial paths. The algorithm computes a lower bound and an upper bound for each partial path in the tree.

A partial path is prioritized to be branched if the associated lower bound is the minimum among **all** partial paths of **all** BnB trees. For example, in Figure 3(b), we assume that the partial path between k^+ and W_1 has the minimum lower bound among all the partial paths that have been found so far in all the 12 BnB trees. This partial path is branched, resulting in 4 new partial paths (Figure 3(c)). To speed up the search, the algorithm prunes branches in BnB trees if the expansion of the branches would not lead to better results than expanding the most promising branch found so far. The algorithm maintains a global minimum upper bound, which may decrease as more partial paths are expanded. When the global minimum upper bound becomes lower than the lower bound of a partial path, the algorithm will dismiss (prune) the partial path in the future.

The lower bound or upper bound of a partial path is a numeric value that is aggregated from two parts. The first part is the travel cost of the partial path, i.e., the cost of travelling from the root of the BnB tree to the last node on the path. The second part is an estimate of the travel cost associated with the unvisited nodes. The lower bound and the upper bound differ in the way to compute the second part. For the lower bound, the second part is the sum of the one-hop distances between each unvisited node and its nearest neighbor, even if multiple unvisited nodes share the same nearest neighbor. The lower bound is likely to be lower than the actual delivery cost because it only concerns one-hop neighbors rather than a complete path when estimating the travel cost in the second part. The upper bound, on the other hand, computes the second part by assuming that the driver will follow a Hamiltonian path for the rest of the trip. The Hamiltonian path is computed with the nearest-neighbor-based routing algorithm described earlier (Section 4.2.1).

4.3 Detour Control

The methods presented in Section 4.1 and Section 4.2 can help reduce delivery costs. However, it is still possible that some ad hoc drivers need to make large detours. We present two methods that guarantee the maximum detour distance by controlling the size and shape of the area where an ad hoc driver can pick up and deliver goods. These methods can be used to set the limit of the service cost SC. The first method limits the maximum detour for visiting individual stores or customers. This method can be applied in the assignment stage. The second method enables a more aggressive control by setting a distance threshold for the whole delivery route. This method can be applied in the routing stage. Due to the constraints imposed by these detour control methods, the proportion of customers served by ad hoc drivers (**PC** as defined in Section 3) may decrease when the methods are applied. The customers who cannot be served by ad hoc drivers need to be assigned to dedicated drivers. Our experiments show that PC can still be close to 100% when the first detour control method is applied.

4.3.1 *Path-Expansion Method.*

This method controls the maximum detour for visiting individual stores or customers. To achieve this, the method finds an area covering the driver's original route. The maximum distance from any point on the boundary of the area to any node on the driver's original route is $f \times dist_k$, where f is a **detour fraction** and $dist_k$ is the length of the original route. With this restriction in place, a driver can only visit pickup stores and customers within that area. We show an example of this method in Figure 4, where the fraction parameter is set to 0.5. The figure shows a search space centred at each node on the original route. The distance between any point on the boundary of a search space and the centre of the space is less or equal to 0.5 times the original route's distance. The whole serviceable area, where the driver can pickup and deliver goods, is the union of all the search spaces. We should note that the search spaces are not perfect circles as the distances are network distances rather than Euclidean distances.

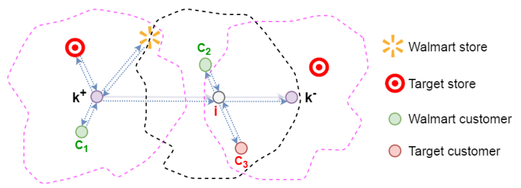

Figure 4: The search space expands from driver k's route by up to $f = 0.5$ times the route's distance. The expansion is made from each node on the original route (k^+, i, k^-). Walmart and Target stores and some customers are within the expanded area. The delivery route (shown in blue dotted arrows) includes round trips to the stores and the customers in this example.

This method enforces an upper bound of the delivery cost. Let f be the detour fraction, $dist_k$ be the distance of the original route from k^+ to k^-, $|R_k|$ be the number of retailers in the expanded area and $|C_k|$ be the number of customers in the expanded area. We have the following proposition.

PROPOSITION 1. *The upper bound of the driver's delivery cost $\mathcal{U}_k$ can be calculated as:*

$$\mathcal{U}_k = \big(2f(|R_k| + |C_k|) + 1\big)dist_k \tag{2}$$

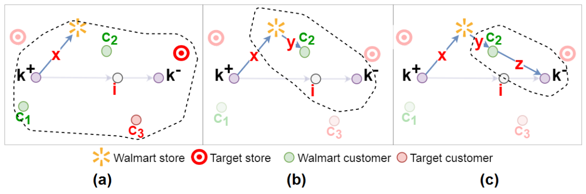

Figure 5: (a) An ellipse with constant $t \times dist_k$ where t is the maximum cost threshold and $dist_k$ is the distance of the driver's original route. The driver's source (k^+) and destination (k^-) are the two foci. The Walmart store is added to the partial route at this step. (b) In the next step, a new ellipse is created with foci Walmart and k^-. The ellipse's constant is $t \times dist_k - x$, where x is the network distance from k^+ to Walmart. Customer c_2 is added to the partial route. (c) In the last step, a new ellipse is created with foci c_2 and k^-. The ellipse's constant is $t \times dist_k - x - y$, where y is the network distance from Walmart to c_2. The complete route is found at this step.

Proof. In the worst case, any store or customer is only connected with a specific node on the driver's original route, which means the driver must diverge from a point on the original route, visit the store or customer, then return to the diverging point before going forward along the original route. Therefore, we can conclude that the cost of visiting a store or customer is at most twice of $f \times dist_k$ due to the round-trip. Assuming the driver needs to visit all the retailers and the customers in the area, the maximum detour distance would be $2f(|R_k| + |C_k|)dist_k$. Finally, we must also count the distance of the driver's original route as the original route is included in the delivery route. □

4.3.2 Cost-Threshold Method. The cost upper bound achieved by the path-expansion method depends on the number of retailers and the number of customers. Since these numbers may not be predictable, the upper bound may change in various scenarios. To achieve a stronger cost guarantee, it would be ideal to have a fixed upper bound regardless of the random factors. We develop a method that sets a fixed **maximum cost threshold** t for the entire delivery route. This method controls the size and shape of the area where a delivery route is searched. By applying this method, the upper bound of the delivery cost for a driver k is $t \times dist_k$ where $dist_k$ is the distance of the original route of the driver.

The area for searching a delivery route is located based on the concept of ellipses. An ellipse has two fixed points called foci. For any point in the interior of an ellipse, the combined distance between the point and the two foci is not larger than a constant. In our case, the area where a delivery route is searched can resemble an ellipse. If a driver only ever needs to visit one store or one customer, we can find the whole delivery route with only one ellipse, where the foci are the driver's source and destination and the constant associated with the ellipse is $t \times dist_k$. However, a driver needs to make at least two stops (one store and one customer) during a delivery trip. In some cases a driver needs to make a large number of stops during the trip. The total travel cost of the trip needs to be below the upper bound no matter how many stops are made. For this reason, we need to create a new ellipse whenever a new waypoint is added to a partial route, i.e., when a nearest neighbor is found in the nearest-neighbor-based routing or when a node is branched in the branch-and-bound routing. In the new ellipse, the first focal point is the last node on the partial route, the second focal point remains at the destination of the driver. The new ellipse also needs to be associated with a lower constant because the partial route becomes longer while the maximum cost threshold t is fixed. Specifically, the new constant is the old constant minus the distance of the last leg on the partial route. With the new ellipse, we continue finding the route towards the destination. We repeat this procedure until the destination is reached. This method provides a stronger constraint on service cost (*SC*) than the path-expansion method but it can lead to a lower proportion of customers served by ad hoc drivers (*PC*) due to the smaller space for finding a route.

The procedure of finding delivery routes based on ellipses can be explained with the example in Figure 5.

Table 1: Four variants of our solution based on CD-CRSS model. IRNN-BnB is the default solution when comparing CD-CRSS against other models.

Acronym	Assignment Method	Routing Method
NN−NN	NN	NN
NN−BnB	NN	BnB
IRNN−NN	IRNN with Predecessors	NN
IRNN−BnB	IRNN with Predecessors	BnB

5 EXPERIMENTS

We ran experiments to compare the performance of our CD-CRSS model against the existing model adopted by major retailers. For the existing model, we assume that the goods purchased by a customer are always picked up from the store that is the closest one to the customer and the driver who delivers the goods is the closest driver to the pickup store. For CD-CRSS, we implement four variants of our solution (Table 1). The default solution of CD-CRSS is named *IRNN−BnB* because it uses IRNN with predecessors for assignment (Section 4.1.2) and the branch-and-bound approach for routing (Section 4.2.2). In addition, the experiments include a Mixed-Integer Linear Program (MILP)-based solution that gives the optimal delivery routes.

The effects of different parameters on the performance of the models are evaluated through experiments. *Number of customers* is the total number of customers that submit delivery requests (the size of set C defined in Section 3). Some or all of the customers can be served by ad hoc drivers in different scenarios. *Ratio of customers to drivers* is the ratio of the *number of customers* to the total number of drivers. This parameter controls the number of drivers (the size of set D defined in Section 3). The *maximum degree of MST* is the highest degree of the MST tree that is used for balancing workload between drivers (Section 4.1.2). It is the largest possible number of customers assigned to any driver when CD-CRSS' assignment stage uses the IRNN-based method. *Detour fraction* controls the maximum detour distance for visiting an individual customer or store. It is computed as the ratio of the maximum detour distance to the distance of the original route of a driver. When testing the effects of this parameter, CD-CRSS-based solutions use the path-expansion method for detour control (Section 4.3.1). *Maximum cost threshold* is associated with the fixed upper bound of delivery cost. When

Table 2: Parameter settings

Parameter	Values	Default
Number of customers	8 - 2048	256
Ratio of customers to drivers	1, 2, 4, 8	4
Maximum degree of MST	4, 6, 8, 10, 12	8
Detour fraction	0.05, 0.1, 0.15, 0.2	infinity
Maximum cost threshold	1.05, 1.1, 1.15, 1.2	infinity

testing the effects of this parameter, CD-CRSS-based solutions use the cost-threshold method for detour control (Section 4.3.2). The parameter settings are detailed in Table 2.

A batch of 256 customers as default is shown to be enough in Section 5.1.1. Having a default proportion of 1:4 between ad hoc drivers and customers is also a reasonable assumption, especially because of the pandemic, where the number of customers at home increased drastically. For each set of parameter settings, we run 50 experiments on a random $100km^2$ area in Melbourne. The road network and the store locations of four major retailers in Melbourne are extracted from *OpenStreetMap* [28]. Customer locations are uniformly distributed at random. We simulate ad hoc drivers leaving their workplaces and going home. The start locations of drivers are Zipfian distributed, which mimics crowded areas such as the CBD. The end location of drivers are uniformly distributed at random. The distribution of the results is shown through box plots.

We evaluate the performance based on **service cost (SC)**, **processing time**, **average detour**, and **proportion of served customers (PC)**. *Service cost* is the total travel distance of the ad hoc drivers. *Processing time* is the length of the period in which all the requests are processed. *Average detour* is the average ratio of delivery distance to the distance of the original route. *Proportion of served customers* is the ratio of customers served by ad hoc drivers to all the customers. When comparing CD-CRSS against other models, we also use two other measurements. The first measurement is **service cost ratio**, which is the SC of CD-CRSS divided by the SC of another model. The second measurement is **processing time ratio**, which is the processing time of CD-CRSS divided by the processing time of another model.

5.1 Results

5.1.1 Number of Customers. As shown in Figure 6(a), the service cost achieved by CD-CRSS is up to 20% less than the existing model's service cost. The advantage of CD-CRSS becomes more significant when there are more customers. This is due to the fact that **the benefit of optimizing store selection tends to increase when more customers can get goods from the same store**. The results also show that CD-CRSS is readily deployable since our solution can process 256 requests in only 8 seconds (Figure 6(b)) while a recent study shows that in average 256 delivery requests are submitted for every 12 minutes in Melbourne [27]. Even with 2048 requests, CD-CRSS can complete the process in 100 seconds, which means our solution is suitable for a scenario, where approximately every household in a city of size 5 million puts an online order every day for crowdsourced delivery[2].

[2]It is also noteworthy to mention that unlike processing times in computing, travel gains at the level of 20% are seen as high gains in transport engineering.

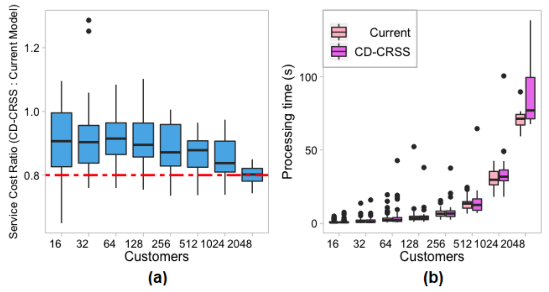

Figure 6: (a) Ratio of CD-CRSS' service cost to the existing model's service cost drops to 80% (red dashed line) when there are 2048 customers. (b) CD-CRSS and the existing model take comparable processing times.

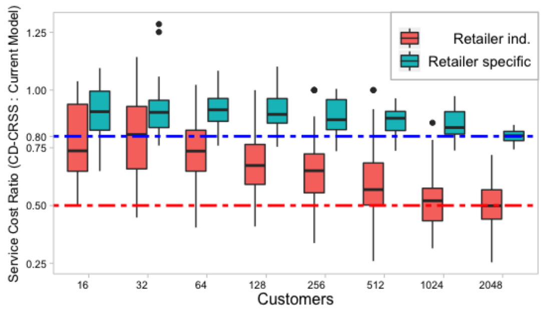

Figure 7: The retailer-specific version of CD-CRSS achieves up to 20% service cost reduction from the existing model (indicated by the blue dashed line). The retailer-independent version of CD-CRSS achieves up to 50% service cost reduction from the existing model (indicated by the red dashed line).

The performance of CD-CRSS can be improved even further if customers can buy the same products from different retailers, i.e., customers buy from a retailer-independent platform like Amazon. By adopting the Amazon-like model, the pickup stores can be selected from a wider range, increasing the possibility to reduce delivery costs further. We conduct an experiment to compare CD-CRSS' performance in the retailer-specific scenario against the model's performance in the retailer-independent scenario (Figure 7). The retailer-independent version of CD-CRSS achieves up to 50% cost reduction from the existing model used by major retailers, which is an exceptionally high value of cost savings in travel.

5.1.2 Customer-Driver Ratio. With a larger ratio of customers to drivers, a driver tends to serve more customers. This can result in the increase of delivery costs due to the increase of detour distance. This is evidenced in Figure 8, where we compare the average detour achieved by four variants of our solution based on CD-CRSS. We can observe that the average detour nearly doubles when the ratio increases from 1 to 8. The default solution (IRNN-BnB) achieves lower detour than other solutions in all the cases. This indicates that the combination of IRNN-based assignment and BnB-based routing yields the best results. The result does not show the detour achieved by NN-BnB when the ratio is 8 because the solution requires prohibitive computation time under the setting.

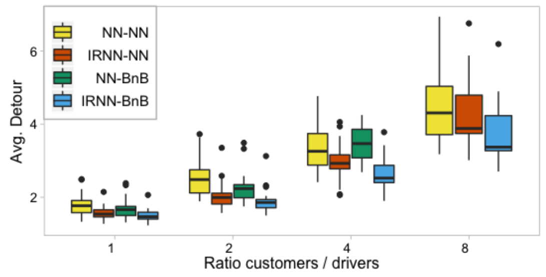

Figure 8: Average detour increases as the ratio of customers to ad hoc drivers increases.

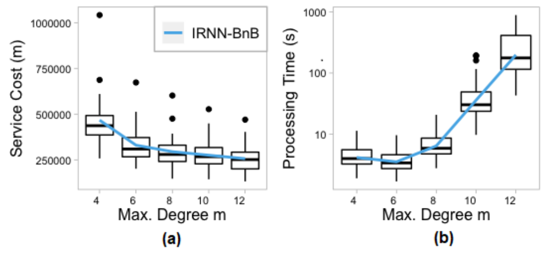

Figure 9: (a) Service cost of IRNN-BnB decreases gradually when workload becomes increasingly unbalanced (the maximum degree m of MST increases). (b) Processing time of delivery plans increases exponentially at the same time.

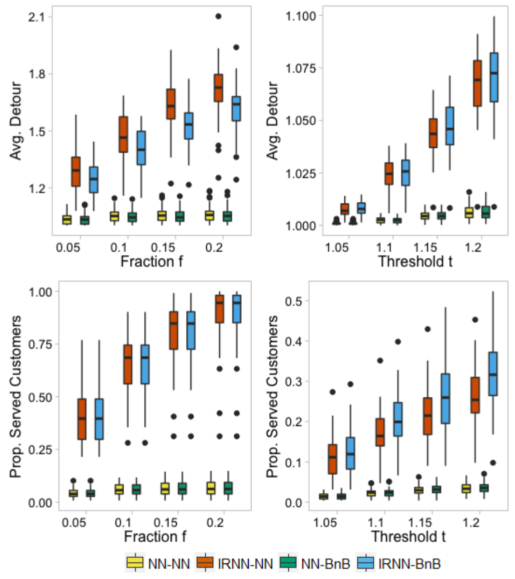

Figure 10: Performance of detour control methods in terms of average detour and proportion of served customers. The left column shows the performance of the path-expansion method with different settings of detour fraction. The right column shows the performance of the cost-threshold method with different settings of the maximum cost threshold.

5.1.3 Maximum Degree of MST. In this experiment, we evaluate the impact of load balancing on the default solution of CD-CRSS, IRNN-BnB. Specifically, we evaluate the effects of the maximum degree m of MST, which determines the aggressiveness of load balancing, on service cost and processing time. Figure 9(a) shows that the service cost decreases gradually with the increase of m. A higher m leads to a more unbalanced workload between drivers, reducing the service cost as the customers can be served in a less number of delivery trips. However, the processing time of delivery plans increases exponentially when m increases (Figure 9(b)). The significant increase of processing time is due to the fact that the number of partial paths that need to be checked in BnB routing grows exponentially. Since we are also after an assignment size that allows efficient computation without significant impact on the service cost, we chose $m = 8$ as the default maximum degree m of MST for IRNN-based assignment in CD-CRSS.

5.1.4 Detour Control Parameters. In the ideal situation, crowdsourced delivery can serve a large proportion of customers using ad hoc drivers while limiting the detours made by the drivers. We presented two detour control methods in Section 4.3, a path-expansion method and a cost-threshold method. The first method applies a detour fraction when expanding the search area of customers and stores. The second method imposes a fixed cost threshold for the entire delivery route. This experiment evaluates the effectiveness of the two methods. The results are shown in Figure 10. Based on the results, the two solutions that use Nearest Neighbor-based assignment (NN-NN and NN-BnB) are not usable when detour control is applied due to the fact that less than 10% of the customers are assigned to ad hoc drivers. We focus on the other two solutions that use IRNN-based assignment in the rest of the section.

For both detour control methods, the average detour achieved by IRNN-NN and IRNN-BnB increases when the upper bound of delivery cost increases. This is shown in the top row of Figure 10. However, the cost-threshold method (the top-right sub-figure) achieves lower average detour values than the path-expansion method, where the cost upper bound is more relaxed (the top-left sub-figure).

We also observe a difference between IRNN-NN and IRNN-BnB in terms of the proportion of customers served by ad hoc drivers. As shown in the bottom-left sub-figure in Figure 10, nearly all customers are served by ad hoc drivers when the detour fraction f is between 0.15 and 0.2 with the path-expansion method. On the contrary, even though the cost-threshold method achieves smaller detours, less than 40% of the customers are assigned to ad hoc drivers (the bottom-right sub-figure). This means that dedicated drivers have to be deployed to serve a large portion of the customers, which can be costly to the retailer. This may negate the benefit of the cost-threshold method. To allow for higher rates of crowdsourcing, we thus recommend using IRNN-BnB with f being as high as possible (acceptable to the ad hoc drivers).

5.1.5 MILP. We also benchmark the default solution of CD-CRSS, $IRNN-BnB$, against the MILP-based solution (Figure 11). For this experiment, the numbers of customers are smaller than those used in other experiments as the MILP solver takes a significantly long time to compute optimal delivery routes with more than 16 customers. Also, we do not perform load balancing for the drivers in $IRNN-BnB$ due to the low number of customers in this experiment.

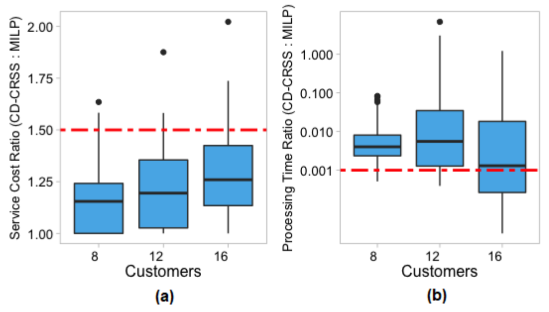

Figure 11: (a) The service cost of the default CD-CRSS solution, *IRNN–BnB*, is less than 1.5 times the optimal service cost computed by the MILP solver. (b) The CD-CRSS solution runs 2-3 orders of magnitude faster than the MILP solver.

IRNN–BnB yields a service cost close to the cost yielded by the MILP solver (Figure 11(a)). However, our solution runs two to three orders of magnitude faster than the MILP solver (Figure 11(b)).

6 CONCLUSION

Our work has shown that it is possible to reduce the cost of crowdsourced delivery through concurrent selection of pickup stores and drivers. Our solution is highly effective in terms of cost saving. It can reduce delivery costs by up to 50% compared with the existing model adopted by major retailers. At the same time, our solution can process once-a-day delivery for every household in a city with millions of people as it can process over 2000 requests in a batch in 100 seconds. Future research can investigate the possibility of dynamic adjustments to delivery routes as our work assumes drivers would not change routes once the routes have been computed.

REFERENCES

[1] Agatz, Niels and Erera, Alan L and Savelsbergh, Martin WP and Wang, Xing. 2011. Dynamic ride-sharing: A simulation study in metro Atlanta. *Procedia-Social and Behavioral Sciences* 17 (2011), 532–550.

[2] Archetti, Claudia and Guerriero, Francesca and Macrina, Giusy. 2021. The online vehicle routing problem with occasional drivers. *Computers & Operations Research* 127 (2021), 105144.

[3] Archetti, Claudia and Savelsbergh, Martin and Speranza, M Grazia. 2016. The vehicle routing problem with occasional drivers. *European Journal of Operational Research* 254, 2 (2016), 472–480.

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

[5] Cao, Bin and Alarabi, Louai and Mokbel, Mohamed F and Basalamah, Anas. 2015. Sharek: A scalable dynamic ride sharing system. In *MDM*, Vol. 1. IEEE, 4–13.

[6] Chen, Ping and Chankov, Stanislav Milkov. 2017. Crowdsourced delivery for last-mile distribution: An agent-based modelling and simulation approach. In *2017 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*. IEEE, 1271–1275.

[7] Chen, Wenyi and Mes, Martijn and Schutten, Marco. 2018. Multi-hop driver-parcel matching problem with time windows. *Flexible Services and Manufacturing Journal* 30, 3 (2018), 517–553.

[8] Chen, Zaiben and Shen, Heng Tao and Zhou, Xiaofang and Yu, Jeffrey Xu. 2009. Monitoring path nearest neighbor in road networks. In *SIGMOD*. 591–602.

[9] Peng Cheng, Xiang Lian, Lei Chen, and Cyrus Shahabi. 2017. Prediction-based task assignment in spatial crowdsourcing. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 997–1008.

[10] Camila F Costa and Mario A Nascimento. 2018. In-route task selection in crowdsourcing. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 524–527.

[11] Iman Dayarian and Martin Savelsbergh. 2020. Crowdshipping and same-day delivery: Employing in-store customers to deliver online orders. *Production and Operations Management* 29, 9 (2020), 2153–2174.

[12] Dingxiong Deng, Cyrus Shahabi, and Linhong Zhu. 2015. Task matching and scheduling for multiple workers in spatial crowdsourcing. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 1–10.

[13] Escudero, Laureano F. 1988. An inexact algorithm for the sequential ordering problem. *European Journal of Operational Research* 37, 2 (1988), 236–249.

[14] Katarzyna Gdowska, Ana Viana, and João Pedro Pedroso. 2018. Stochastic last-mile delivery with crowdshipping. *Transportation research procedia* 30 (2018), 90–100.

[15] Hong, Jinseok and Lee, Minyoung and Cheong, Taesu and Lee, Hong Chul. 2019. Routing for an on-demand logistics service. *Transportation Research Part C: Emerging Technologies* 103 (2019), 328–351.

[16] Huang, Yan and Bastani, Favyen and Jin, Ruoming and Wang, Xiaoyang Sean. 2014. Large scale real-time ridesharing with service guarantee on road networks. *VLDB* 7, 14 (2014), 2017–2028.

[17] Fariha Tabassum Islam, Tanzima Hashem, and Rifat Shahriyar. 2021. A Privacy-Enhanced and Personalized Safe Route Planner with Crowdsourced Data and Computation. In *ICDE*. IEEE, 229–240.

[18] Jensen, Christian S and Kolářvr, Jan and Pedersen, Torben Bach and Timko, Igor. 2003. Nearest neighbor queries in road networks. In *Proceedings of the 11th ACM International Symposium on Advances in Geographic Information Systems (GIS)*. 1–8.

[19] Kafle, Nabin and Zou, Bo and Lin, Jane. 2017. Design and modeling of a crowdsource-enabled system for urban parcel relay and delivery. *Transportation Research Part B: Methodological* 99 (2017), 62–82.

[20] Karp, Richard M. 1972. Reducibility among combinatorial problems. In *Complexity of Computer Computations*. Springer, 85–103.

[21] Yafei Li, Ji Wan, Rui Chen, Jianliang Xu, Xiaoyi Fu, Hongyan Gu, Pei Lv, and Mingliang Xu. 2021. Top-kk Vehicle Matching in Social Ridesharing: A Price-Aware Approach. *TKDE* 33, 3 (2021), 1251–1263.

[22] Lu, Songjian and Lu, Xinghua. 2013. An exact algorithm with the time complexity of $O^*(1.299^m)$ for the weighed mutually exclusive set cover problem. *arXiv preprint arXiv:1302.5820* (2013).

[23] Luo, Hui and Bao, Zhifeng and Choudhury, Farhana and Culpepper, Shane. 2019. Dynamic Ridesharing in Peak Travel Periods. *TKDE* (2019), 1–1.

[24] Ma, Shuo and Zheng, Yu and Wolfson, Ouri. 2013. T-share: A large-scale dynamic taxi ridesharing service. In *ICDE*. 410–421.

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

[26] Macrina, Giusy and Pugliese, Luigi Di Puglia and Guerriero, Francesca and Laporte, Gilbert. 2020. Crowd-shipping with time windows and transshipment nodes. *Computers & Operations Research* 113 (2020), 104806.

[27] Morgan, Roy. 2019. Over 5 million Australians consider buying groceries online. http://www.roymorgan.com/findings/7911-australian-online-grocery-shopping-march-2019-201903220623. [Online; accessed 09-May-2021].

[28] OpenStreetMap Contributors. 2017. Planet dump retrieved from https://planet.osm.org . https://www.openstreetmap.org. [Online; accessed 19-January-2018].

[29] Papadias, Dimitris and Zhang, Jun and Mamoulis, Nikos and Tao, Yufei. 2003. Query processing in spatial network databases. In *VLDB*. Elsevier, 802–813.

[30] Raviv, Tal and Tenzer, Eyal Z. 2018. Crowd-shipping of small parcels in a physical internet. https://www.researchgate.net/publication/326319843_Crowd-shipping_of_small_parcels_in_a_physical_internet. [Online; accessed 09-May-2021].

[31] Marc Sevaux and Kenneth Sörensen. 2008. Hamiltonian paths in large clustered routing problems. In *Workshop on Metaheuristics for Logistics and Vehicle Routing, EU/ME'08*. Troyes, France, 411–417.

[32] Shekhar, Shashi and Yoo, Jin Soung. 2003. Processing in-route nearest neighbor queries: a comparison of alternative approaches. In *Proceedings of the 11th ACM International Symposium on Advances in Geographic Information Systems (GIS)*. 9–16.

[33] Tong, Yongxin and Zeng, Yuxiang and Zhou, Zimu and Chen, Lei and Ye, Jieping and Xu, Ke. 2018. A unified approach to route planning for shared mobility. *VLDB* 11, 11 (2018), 1633–1646.

[34] Yıldız, Barış. 2021. Express package routing problem with occasional couriers. *Transportation Research Part C: Emerging Technologies* 123 (2021), 102994.

[35] Yoo, Jin Soung and Shekhar, Shashi. 2005. In-route nearest neighbor queries. *GeoInformatica* 9, 2 (2005), 117–137.

[36] Yan Zhao, Kai Zheng, Jiannan Guo, Bin Yang, Torben Bach Pedersen, and Christian S Jensen. 2021. Fairness-aware Task Assignment in Spatial Crowdsourcing: Game-Theoretic Approaches. In *ICDE*. IEEE, 265–276.

[37] Zheng, Bolong and Huang, Chenze and Jensen, Christian S and Chen, Lu and Hung, Nguyen Quoc Viet and Liu, Guanfeng and Li, Guohui and Zheng, Kai. 2020. Online Trichromatic Pickup and Delivery Scheduling in Spatial Crowdsourcing. In *ICDE*. 973–984.