Crowdsourced Delivery: A Dynamic Pickup and Delivery Problem with Ad-hoc drivers

Alp M. Arslan, Niels Agatz, Leo Kroon, Rob Zuidwijk

Rotterdam School of Management, Erasmus University, The Netherlands

*E-mail: arslan@rsm.nl

September 5, 2016

The trend towards shorter delivery lead-times reduces operational efficiency and increases transportation costs for internet retailers. Mobile technology, however, creates new opportunities to organize the last-mile. In this paper, we study the concept of crowdsourced delivery that aims to use excess capacity on journeys that already take place to make deliveries. We consider a peer-to-peer platform that automatically creates matches between parcel delivery tasks and ad-hoc drivers. The platform also operates a fleet of backup vehicles to serve the tasks that cannot be served by the ad-hoc drivers. The matching of tasks, drivers and backup vehicles gives rise to a new variant of the dynamic pick-up and delivery problem. We propose a rolling horizon framework and develop an exact solution approach to solve the various subproblems. In order to investigate the potential benefit of crowdsourced delivery, we conduct a wide range of computational experiments. The experiments provide insights into the viability of crowdsourced delivery under various assumptions about the environment and the behavior of the ad-hoc drivers. The results suggest that the use of ad-hoc drivers has the potential to make the last-mile more cost-efficient and can reduce the system-wide vehicle-miles.

1 Introduction

Despite the spectacular growth of online sales, internet retailers still face many logistical challenges in the successful fulfilment of goods ordered online. One of the main challenges is to provide convenient home delivery services in a cost-efficient way. The recent trend towards shorter delivery lead-times and same-day delivery further increases the strain on transport efficiency. At the same time, mobile internet technology gives rise to new opportunities to organize the last-mile. One of those new opportunities is *crowdsourced* delivery. This concept entails the use of excess capacity on journeys that already take place to support delivery operations. By using existing traffic flows, this could potentially enable faster and cheaper deliveries. Moreover, it reduces the negative environmental impact, such as emissions, of the use of dedicated delivery vehicles. This development is part of a bigger trend that is called the "sharing economy" which allows people to enhance the use of resources through the redistribution, sharing and reuse of excess capacity in goods and services.

In 2013, the retailer Walmart announced that it was investigating the use of its in-store customers to deliver goods to its online customers on their way home from the store. In the same year, DHL ran a pilot in Stockholm called 'MyWays' using ordinary people to perform some of their deliveries (Morphy 2014). In a similar vein, Amazon recently launched a service called Amazon Flex in Seattle that supports the use of self-employed drivers to deliver packages for them.

In recent years, we have seen the advent of peer-to-peer (P2P) market places for transportation. Some of these platforms focus on long distance shipping (Friendshippr, Roadie), while others focus on (on-demand) local delivery services (Kanga, Renren Kuaidi, Deliv, Trunkrs and Amazon flex). All of these companies offer online platforms and mobile smartphone apps to quickly connect delivery tasks (parcels that need to be shipped) and drivers willing to make a delivery along their route (see Table 1 for an overview). The drivers pick up parcels from a retail store, warehouse or dedicated pickup location, and deliver them to customer locations on their way home or to work.

Instead of traditional employees or service providers, the drivers act voluntarily on their

Table 1: Examples of Crowd-Delivery Platforms that offer Same-day Delivery (July 2016)

Name	Compensation scheme	Information from ad-hoc drivers	Where
Deliv	Hourly rate	Time period	17 U.S. cities
Renren Kuaidi	Per package	Time period	16 Chinese cities
Trunkrs	Per package	Time, origin and destination	The Netherlands
Kanga	Hourly rate	Time-period	1 U.S. city
Amazon flex	Hourly rate	Time-period	17 U.S. cities

own initiative. They are willing to make deliveries along their route to help others, support environmentally friendly deliveries, and potentially earn some extra money. In particular, drivers are willing to take a parcel along a specific journey that they are already making. This is different from systems in which the drivers only perform deliveries to earn money. In this setting, drivers may vary greatly with respect to their time and detour flexibility. Some drivers may only want to make a small detour to take a parcel on a trip that they were already making, others may be willing to make multiple deliveries. When each driver can be matched with at most one delivery task, we can model the problem as a bipartite matching problem (Agatz et al. 2011). However, if we want to allow multiple pickups and drop-offs in a single trip, we also need to consider the route sequence, which makes the problem more challenging.

To ensure that all parcels are delivered in time, a P2P delivery platform may use a third-party service to deliver the tasks for which no ad-hoc driver could be found (e.g. Dutch startup PickThisUp uses this model). Moreover, to ensure the reliability and trustworthiness of the ad-hoc drivers, it could use various feedback mechanisms and external regulations (see Einav et al. (2015) for an overview of P2P trust generating mechanisms). Most of the current platforms let participants rate the drivers in terms of their reliability and effectiveness. Deliv, for example, states that it "only maintains driver partnerships with those drivers who have a consistent record of timeliness, reliability, and good overall delivery results." Several others also check their drivers in advance by verifying their drivers' license, insurance and registration, doing background checks and reviewing their driving history.

In this paper, we focus on a local P2P delivery platform that automatically matches delivery tasks and ad-hoc drivers to facilitate on-demand delivery. The platform also operates

a set of dedicated back-up vehicles to serve tasks for which the use of an available ad-hoc driver is not feasible or not efficient. As such, the crowdsourcing provider needs to assign delivery tasks to ad-hoc drivers and backup vehicles and determine the associated delivery routes. We consider a same-day delivery setting in which both tasks and drivers dynamically arrive over time.

The main contributions of this paper are as follows: firstly, we introduce and describe a new route planning problem that involves the use of ad-hoc drivers and dedicated backup vehicles to perform on-demand deliveries. We present a rolling horizon framework and develop an exact solution approach (based on a matching formulation) to repeatedly solve the various versions of the off-line problem. Secondly, we conduct an extensive computational study to investigate under what circumstances it is viable to use crowdsourced transportation to enable on-demand deliveries. To quantify the benefits, we compare the performance of a crowdsourced system with a traditional dedicated delivery system. The results indicate that the use of ad-hoc drivers can significantly reduce transportation costs.

The remainder of the paper is organized as follows: we discuss the relevant literature in the next section. In section 3, we formally describe the problem. In section 4, we explain the implementation of our rolling horizon framework and formulate the problem. In section 5 we provide a solution approach for the routing subproblem. In section 6, we describe our instances and present the results from our numerical experiments. Finally, in section 7 we provide some concluding remarks and directions for future research.

2 Related literature

Thus far, most research in the area of crowd-sourcing has focussed on *virtual* tasks that can be done remotely over the internet such as text editing, translation and debugging (see e.g. Doan et al. (2011)). A recently emerging area of research considers the idea of using the crowd to conduct physical tasks such as parcel delivery (Suh et al. 2012, Sadilek et al. 2013, Rougès and Montreuil 2014). To effectively organize this, a crowdsourcing provider should assign delivery tasks to ad-hoc drivers and backup vehicles in real-time.

At its core, the crowdsourced delivery problem is a pickup and delivery problem (PDP) that aims to transport goods from origins to destinations at minimum costs. This links our problem to the huge body of literature on PDPs, see Berbeglia et al. (2007) for an overview. Since we consider an on-demand service, our problem is also related to the literature on the dynamic pickup and delivery problems (DPDP) (see Berbeglia et al. (2010)). Our problem is also closely related to the recent work in the context of the same-day delivery of goods ordered online from a single depot (see Klapp et al. (2016) and Voccia et al. (2015)). In the stream of this research, developing strategies for finding the optimal timing for the vehicle departures and optimal assignment of parcels between the vehicles are the main challenges Savelsbergh and Van Woensel (2016).

Unlike the traditional PDP setting, we only use a dedicated fleet of vehicles as a backup option for the independent ad-hoc drivers. In that sense, our problem is similar to ride-sharing or carpooling where individual travelers share a ride to save on their travel costs by using their own vehicles (Furuhata et al. 2013, Agatz et al. 2012). A recent study by Agatz et al. (2011) investigates the viability of dynamic ride-sharing in which trips are announced shortly before departure. The authors create single rider, single driver ride-share matches and propose a rolling horizon approach for dealing with real-time updates. The study shows that the success of a ride-sharing system depends on a sufficiently large number of participants. To guarantee a certain service level to the riders, the ride-share service provider could use (a small number of) dedicated drivers to serve riders that would otherwise remain unmatched. Lee and Savelsbergh (2015) investigate how many of such dedicated drivers are needed to achieve a certain service level. They formulate the problem as an integer program and present a heuristic approach to solve realistic-size instances. In a similar vein, Stiglic et al. (2015) explore the benefits of using meeting points to improve the performance of a ride-sharing system. When drivers are willing to walk to and from a meeting point, this may allow drivers to carry multiple riders without the inconvenience of many additional stops. Baldacci et al. (2004) describe a static car-pooling problem that aims to assign a set of drivers to riders. Similar to our paper, drivers can do multiple pickups along their routes. In contrast to our problem, they assume a simplified routing structure

in which all riders and drivers have the same destination (i.e. the workplace) and consider a static problem setting in which all requests are known in advance. They use maximum ride time restrictions to ensure the convenience of the passengers. The authors formulate the problem as a set-partitioning problem and propose an exact solution method based on column generation.

Several recent papers study the use of existing traffic flows to enable freight transportation. Li et al. (2014) and Li et al. (2016) consider a setting in which taxis transport parcels along with their passengers. Both papers propose heuristic solution strategies to insert parcel requests into existing taxi routes. Depending on the flexibility of the passengers, a taxi may stop multiple times to pick up or drop-off a parcel. In a similar vein, Ghilas et al. (2013) and Masson et al. (2014) explore the potential of using public transportation in parcel transportation. The authors introduce a PDP that aims to synchronize delivery vehicles with the scheduled city buses. In line with these studies, Fatnassi et al. (2015) investigate the integration possibility of passengers and parcels transportation in the context of the automated transport systems for city logistics.

Most similar to our work is the work of Archetti et al. (2016) that analyzes a setting in which occasional drivers complement a traditional delivery service. Similar to our study, the authors aim to minimize the sum of the amount paid to the ad-hoc drivers and the routing cost of the dedicated vehicles. In contrast to our work, they consider a static problem setting without time windows in which the occasional drivers are allowed to make only a single delivery. Archetti et al. present a heuristic solution approach that combines variable neighborhood search and tabu search.

3 Problem description

We consider an online crowdsourcing platform that continuously receives new delivery tasks and driver trip announcements over time. Let N denote the set of all origins and destinations, d_{lm} the travel distance, and t_{lm} the travel time between locations $l, m \in N$.

Let P be the set of delivery task (parcel) announcements. Delivery task $p \in P$ has a

pickup location o_p, which can be a retail store, warehouse or dedicated pickup point, and a drop-off location d_p, which is usually the home of the online buyer. The task has an earliest pickup time e_p when it is ready to be picked up and a latest arrival time l_p that corresponds to the time that it has to be delivered. Without loss of generality, we consider a setting in which the parcel needs to be delivered within a certain delivery lead-time L_p, e.g. within 2 hours, where $L_p = l_p - e_p$, $L_p \geq t_{o_p,d_p}$. This is similar to the same-day delivery service model that is used by companies UberRUSH and Shutl. For a given deadline, we can calculate the implied latest departure time $\bar{l_p}$ by $l_p - t_{o_p,d_p}$.

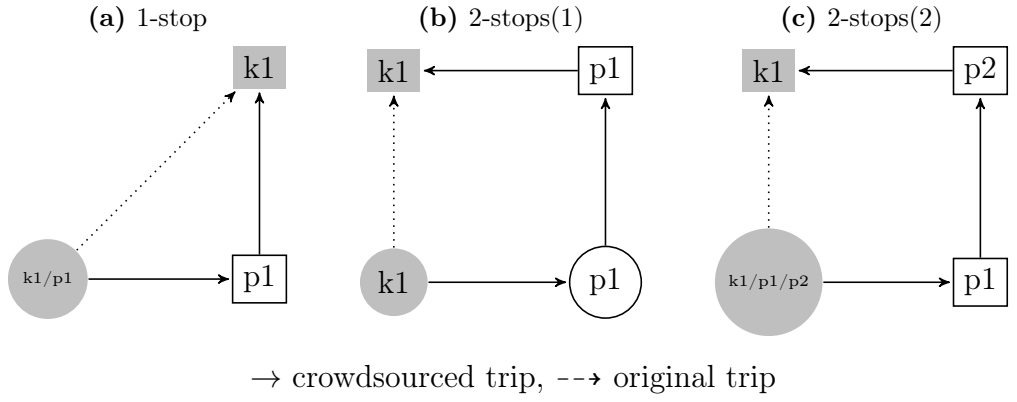
Let K be the set of driver announcements. The driver's trip announcement $k \in K$ specifies his origin o_k and destination d_k. A driver $k \in K$ has an earliest departure time e_k and a latest arrival time l_k. The driver also specifies a maximum travel time, T_k, where $t_{o_k,d_k} \leq T_k \leq l_k - e_k$ and a departure time flexibility, denoted by $F_k = l_k - e_k - t_{o_k,d_k}$.

Besides the detour and departure time flexibility, drivers may also want to specify the maximum number of additional stops that they are willing to make. Let $Q_k \in \mathbb{Z}^+$ denote the *stop willingness* of driver k. Multiple pickups or drop-offs at the same location count as a single stop. As such, the stop willingness restricts the number of different locations that is visited by the driver and therefore reflects the level of inconvenience the ad-hoc driver is willing to accept.

Serving a task is associated with at most two stops: one at the pickup location and one at the drop-off location. When the driver's origin coincides with the pickup location of the task, he needs only one additional stop to make the delivery (See Figure 1a). This corresponds to Walmart's idea to let store customers deliver packages to online buyers along their route from the store to home. Figure 1b shows an example in which the driver's origin is different from the pickup location of the task. In this case, the driver needs to make two additional stops, i.e. one pickup and one drop-off. Another example that requires two additional stops is depicted in Figure 1c where the driver picks up two parcels at his origin and then makes two drop-offs.

To simplify notation, we assume that the time and stop restrictions are more restrictive than the capacity restrictions. This seems like a reasonable assumption as most consumer

Figure 1: A driver(grey) and tasks(white) travelling from his origin (circle) to destination (square)



goods are small enough to easily fit in the trunk of a car (86 percent of Amazon's packages are under 5 pounds and small enough to be shipped even by a drone, (Popper 2015)). To accommodate a setting in which we transport larger objects such as furniture or white goods we could easily introduce an additional constraint on the volume.

We define a *job* j as a set of tasks, where a job can consist of a single task or multiple tasks. The set J denotes the collection of all jobs that are in at least one feasible match. A match (k, j) between driver k and job j is feasible if there exists a *feasible* route r in which the driver starts from his origin o_k, covers all tasks in j and ends at his destination d_k. A route r is *feasible* if it satisfies the following constraints.

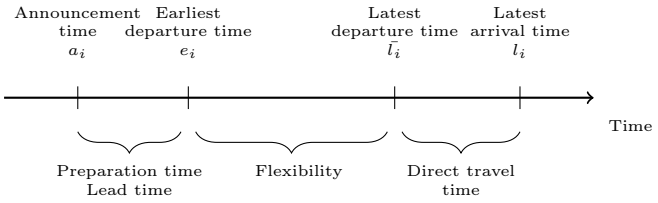
- *Stop constraint.* The number of unique locations visited in route r_{kj} is less than or equal to $Q_k + 2$ (including the origin and destination of the driver).
- *Driving time constraint.* The total travel time of r_{kj} is less than or equal to T_k.
- *Time schedule constraints.* Driver k can not depart before its earliest departure time e_k or arrive after its latest arrival time l_k. Each task $p \in P$ cannot be picked up before its earliest pickup time e_p or arrive after its latest arrival time l_p.
- *Precedence constraints.* For each task $p \in P$, a driver picks the parcel up before dropping it off. This implies that the difference between the drop-off time and the pickup time of task $p \in P$ is greater than or equals to t_{o_p,d_p}

Let R_{kj} be the set of all feasible routes for driver k and job j and R be the set that consists of all feasible routes.

Let B be the set of dedicated backup vehicles. Each vehicle $b \in B$ starts and ends all dispatches from a specific depot and has an earliest departure time e_b and a latest arrival time l_b back at the depot. A match (b, j) between backup vehicle b and job j is feasible if there exists a feasible route r in which a backup vehicle starts from the depot, covers all tasks in j and ends at the depot. Routes for the backup vehicle are feasible if they satisfy the time schedule and precedence constraints. There are no further restrictions on driving time or stops. Table 2 summarizes the main notation.

The system is characterized by the continuous arrival of drivers and tasks. Each driver $k \in K$ is announced to the delivery platform at time $a_k \leq e_k$. We call the time between the announcement time a_k and the earliest departure time e_k the announcement lead-time, $A_k = e_k - a_k$. However for tasks, the announcement lead-time represents the preparation time of a task that is needed to be ready for shipping. The general timeline of a delivery task and a driver can be found in figure 2.

Figure 2: Delivery Tasks and Drivers timeline



We assume that an ad-hoc driver receives a fixed fee for each delivery task plus a per-mile fee for the detour. The costs of using a dedicated backup vehicle depends on the distance driven. The platform earns the difference between the total delivery revenues and the total costs, consisting of the costs of the ad-hoc drivers and the backup vehicle costs. We assume that the platform accepts all delivery tasks and aims to minimize the total system-wide delivery cost.

Table 2: Notation

N	Set of all locations, index i
P	Set of parcel tasks, index p
K	Set of ad-hoc drivers, index k
B	Set of backup drivers, index b
J	Set of all jobs, i.e. combinations of tasks, index j
L_p	Delivery lead-time of task p
T_k	Maximum travel time of driver k
Q_k	Maximum number of additional stops along the route that driver k is willing to make
F_k	Departure time flexibility of driver k

4 Solving the problem

4.1 Rolling horizon approach

Since both delivery tasks and drivers arrive dynamically throughout the day, we use an event-based rolling horizon framework that repeatedly solves the problem of matching tasks to drivers each time t that a new task or driver arrives. At each iteration q of the rolling horizon approach, we determine the matches based on all information that is available to the system at that point in time. In particular, we run the optimization for all active, i.e. known and still available, tasks and drivers.

At time t, task p is *active* if it is not part of a match committed before time t, arrived before t ($a_p \leq t$) and has not expired yet ($\bar{l}_p \geq t$). This is similar for driver k. The drivers and tasks that are associated with a match that is committed at time t are not included in any of the optimization runs after t. A backup driver b is available in each optimization run with an earliest departure time from the depot e_b that depends on earlier job assignments.

Each optimization run results in a number of tentative matches between jobs, ad-hoc drivers and backup vehicles. In principle, we choose to commit these tentative matches as late as possible. However, we also analyse variants where commitments are made early. The late commitments mean that we do not commit to a tentative match before its latest departure time. The latest departure time of a certain tentative match (k, j) is the latest time that driver k can start driving to serve all tasks in j within their time schedules and

then reach his destination on time. This is similar for the backup vehicles. Note that the back-up vehicles start and end their route at the depot.

Next, we describe the offline problem that we solve in each optimization run within our rolling horizon framework based on all available information at time t.

4.2 Offline problem formulation

As in Stiglic et al. (2015), we can model this problem as a matching problem with side-constraints. Let $D = K \cup B$ denote the set of all drivers, i.e. ad-hoc drivers and backup drivers. We create a node for each driver $d \in D$, and a node for each job $j \in J$. An arc between node d and node j represents a feasible match between driver d and job j. The weight of the arc denotes the routing costs of serving job j by driver d.

Let A be the set of all feasible arcs. Let $J_d,\ \ d \in D$ denote the collection of jobs that driver d can serve, and $J_p,\ \ p \in P$ denote the set of jobs that contains task p. Let x_{dj} be the binary decision variable that indicates whether the arc between driver d and job j is in the solution ($x_{dj} = 1$) or not ($x_{dj} = 0$). The coefficient c_{dj} represents the weight of the arc (k, j), which denotes the cost if driver d is assigned the job j. Then, the problem that aims to minimize the total cost can be formulated as follows:

$$\min \sum_{(d,j)\in A} c_{dj} x_{dj} \tag{1}$$

$$\text{s.t} \sum_{j\in J_d} x_{dj} \leq 1 \ \ \forall d \in D, \tag{2}$$

$$\sum_{j\in J_p} \sum_{d\in D} x_{dj} = 1 \ \ \forall p \in P, \tag{3}$$

$$x_{dj} \in \{0, 1\} \ \ \forall (d, j) \in A. \tag{4}$$

Equation (1) is the objective function that aims to minimize the sum of the costs of ad-hoc

driver matches and the backup driver matches. Constraints (2) makes sure that each driver is assigned to at most one job. Constraints (3) make sure that each task is assigned to one of the drivers or a backup vehicle.

When a job j contains only a single task, there exists only one route, which is the origin of the driver and the task followed by the destination of the task and the driver. However, for jobs containing multiple tasks, there might be more than one feasible route. Thus, the determination of the optimal route for corresponding jobs is a subproblem of our matching formulation. The solution approach for this subproblem is the topic of the next section.

5 Solving the routing subproblem

While there does not exist a polynomial time approach to solve our matching problem with side constraints, it is not difficult to solve in practice. It may, however, be quite time consuming to find all feasible jobs (x_{kj} variables). The reason for this is that to assess the feasibility of serving a specific set of tasks, we need to determine the sequence in which to serve them. This means that it involves solving the Traveling Salesman with Time Windows and Precedence Constraints (TSP-TWPC) as a subproblem, see Mingozzi et al. (1997). A description of TSP-TWPC can be found in the appendix of this paper. Savelsbergh (1985) showed that finding a feasible solution of TSP-TW is NP-complete. In the worst case, when each driver could serve all tasks in a single trip, the number of feasible matches for p tasks and k drivers is $O(k2^p)$. However, in our problem setting, due to the time and stop restrictions, the number of tasks per job is relatively small, which implies that the number of feasible routes is likely to be far less in practice.

Note that for a new optimization run q, we only have to create new jobs (x_{kj}) for the task or driver that arrived after the previous iteration $q-1$. For all currently active drivers and tasks that were already active in run $q-1$, we only need to check whether the jobs found previously are still time feasible.

5.1 Theoretical insights

Observation 1: A job $j \in J$ does not have a feasible route if there is a subset $j' \subset j$ that has no feasible route (Stiglic et al. 2015).

This observation implies that any *feasible* job for a specific driver is a union of smaller feasible jobs. A match between one driver and two tasks is only feasible if both tasks are individually feasible with this driver. A match between one driver and three tasks is only feasible if all task pairs are also feasible and so forth. Another implication of this observation is that, if there are two tasks that cannot form a feasible job, all unions that include these two tasks are infeasible. We use these two properties to reduce the number of jobs to be considered in our recursive algorithm.

Observation 2: A route r is not feasible if one of the sub-routes $r' \subset r$ is not feasible. A sub-route can be obtained by removing one or more tasks from the original route.

For each feasible job with w tasks, we store each feasible route instead of keeping the best one in our recursive algorithm. For each driver, we use the feasible routes for w tasks to construct the feasible routes with $w + 1$ tasks by iteratively inserting a task, i.e. a pickup and a drop-off, in the route. According to Observation 2, we do not have to consider the route sequences that we found to be infeasible with w tasks.

Observation 3: Any feasible pickup and delivery route r can be transformed to a feasible *clustered route* r', where $\text{dist}(r') \leq \text{dist}(r)$. A *clustered route* is a pick-up and delivery route that does *not* revisit the same pickup location while still carrying tasks that originate from that location.

The proof of this Observation can be found in the appendix

This implies that there exists an optimal *clustered route*, which means that we do not have to consider mixed routes when recursively building the routes in order to find the shortest route.

Observation 4: When picking up several tasks at the same location one after the other, then the pickup sequence has no impact on the routing length.

This means that we can reduce the search space for feasible routes by applying some simple symmetry breaking rules in our recursive algorithm.

5.2 Exact recursive algorithm

Based on these observations, all feasible jobs with respect to the driver $k \in K$ and the backup driver $b \in B$ can be determined by using a recursive algorithm. Naturally, the recursion starts with determining the jobs with just a single task, and combines these single tasks to make jobs of two, three tasks and so on. Let J_k^w be the set of jobs with w tasks that are feasible for driver k. Let R_k be the set of all feasible routes that driver k can make. Then, the recursive algorithm is presented in Algorithm 1.

Algorithm 1 Recursive Algorithm

Precondition: The list J^1 of all feasible pairs of a driver and a single-task job and the set $R_{k,j}$ of associated routes to serve job j with driver k.
Postcondition: All feasible driver-jobs matches

for all $k \in K$ **do**
  w $\leftarrow 2$
  for all $j \in J_k^{w-1}$ **do**
    for all $\{p\} \in J_k^1 \wedge\ p \notin j$ **do**
      $R_{k,j\cup p} \leftarrow \emptyset$
      if SUBFEAS$((j,p,k))$ **then**
        $R_{k,j\cup p} \leftarrow$ FINDROUTES$((R_{k,j},p))$
      end if
      if $R_{k,j\cup p} \neq \emptyset$ **then**
        $R_k \leftarrow R_k \cup R_{k,j\cup p}$
      else
        the job $(j \cup p)$ is infeasible
      end if
    end for
  end for
  if $w < Q_k \wedge J_k^w \neq \emptyset$ **then**
    $w \leftarrow w + 1$
  else
    J_k and R_k are determined. Go to a new driver.
  end if
end for

Algorithm 1 takes the list of all drivers and tasks as an input. It begins with determining all feasible driver and task pairs. For each pair of driver k and single-task job j, the route

set $R_{k,j}$, which contains only a single route $r_{k,p}$, $p \in j$, is created. All single-task jobs that are feasible for driver k are clustered in the set J_k^1 and all routes that are associated to J_k^1 are added to the set R_k.

Next, Algorithm 1 generates feasible jobs and routes for each driver sequentially. For a chosen driver k, the algorithm checks a pair of a job with size $w-1$ and a task by calling Algorithm 2, called as `SUBFEAS`, to examine whether the pair is feasible as a job with size w (see lines 3-15 in Algorithm 2). `SUBFEAS` is an application of **Observation 1**. In other words, `SUBFEAS` returns true/false by only looking at already constructed jobs and their tasks. If `SUBFEAS` returns a true value, which means the job size w is a feasible job candidate. To guarantee the feasibility of the job, at least one feasible route has to be found that driver k can make it. At this point, we use the algorithm 2 .

Algorithm 3, called `FINDROUTES` returns all feasible routes, if exists, by inserting the origin and the destination of the input task into all feasible routes of the input job. Based on **Observation 3** and **Observation 4**, the number of possible insertions can be reduced significantly without losing the optimal solution. At line 7 in Algorithm 3, the additional check prevents creating some redundant routes. This additional check speeds up the system in two ways. Firstly, the algorithm never checks some insertions, which saves time. Secondly, the fewer number of routes are generated per jobs leads to examine fewer number of routes for the following the feasibility of the jobs that consists of more tasks. If `FINDROUTES` returns a non-empty set, this means the pair of the input job and task creates a feasible job for driver k. Else, the examined job is infeasible.

5.3 Heuristic recursive algorithm

For larger instances it may take too much time to find the optimal routes for all jobs, especially the routes that are associated with the backup vehicles. We can design heuristics based on our recursion for the routing subproblem by introducing short-cuts in the generation of routes. In particular, we test the following ideas:

Algorithm 2 SUBFEAS.

Precondition: Job j, task p and driver k
Postcondition: False if at least one subset of j and p is infeasible for driver k, true otherwise

function SUBFEAS(j, p, k)
    $z \leftarrow$ `true`
    for all $j \in J_k^{|j|}$ **do**
        j' $\leftarrow \{(j \setminus \{q\}) \cup \{p\}\}, \ \forall q \in j$
        if $j' \notin J_k^{|j|}$ **then**
            $z \leftarrow$ `false`, **return** z
        end if
    end for
    return z
end function

Algorithm 3 FINDROUTES.

Precondition: Job j, task p and driver k
Postcondition: All feasible routes of job j and task p for driver k

function FINDROUTES($R_{k,j}, p$)
    $z \leftarrow \emptyset$
    $p_1 \leftarrow o_p$
    $p_2 \leftarrow d_p$
    for all $r_{k,j} \in R_{k,j}$ **do**
        for $i \leftarrow 0$ to $|r_{k,j}| - 1$ **do**
            if i `satisfies` **Observation 3** and **Observation 4** **then**
                if `feasible`(i, p_1) **then**
                    for $l \leftarrow i + 1$ to $|r_{k,j}|$ **do**
                        if `feasible`(l, p_2) **then**
                            $z \leftarrow z \cup [r_{j,k} \cup (p_1, p_2)]$
                        end if
                    end for
                end if
            end if
        end for
    end for
    return z
end function

- Store only a limited number of the shortest routes of size w in order to find the routes of size $w + 1$ in the recursion.
- Limit the maximum duration of a route for the backup vehicle.

6 Computational experiments

In this section, we evaluate the performance of our solution approaches and assess the viability of the crowdsourced delivery platform in different settings. Section 6.1 describes the instances that we used in the various experiments. Section 6.2 presents the results of our heuristic approach to determine a hindsight lower bound, and section 6.3 presents our results on the real problem.

All experiments were implemented in C++ and conducted on a 2,7 GHz Intel Core i5 and 8 GB 1867 MHz DDR3 of installed RAM. Gurobi 6.51 was used as an IP solver.

6.1 Instance generation

We generate several instances that represent different task and driver characteristics within a square region with a size of 15 km and a depot for the backup vehicles at the center, i.e. at [7.5,7.5]. In particular, we use three different instance types, which we refer to as geographies, to generate the origins and destinations of the tasks and ad-hoc drivers.

The first geography (g1) is inspired by Walmart and considers a setting in which all tasks and ad-hoc drivers start from one single origin (i.e. store), located in the center of the region, while all destinations are uniformly spread over the region. The second geography (g2) has five different origin locations, one in the center and four randomly selected from the service area. Here, each task and ad-hoc driver originates from one of the five locations, where each location is chosen with equal likelihood. In the last geography (g3), origins and destinations of both tasks and ad-hoc drivers are uniformly distributed over the service area.

The announcement times of tasks (a_p) and ad-hoc drivers (a_k) are drawn from a uniform distribution spanning a ten hours service period. We assume that each ad-hoc driver has

the same *system-wide* departure time flexibility of 20 minutes and a stop willingness of 2 stops. The announcement lead-time A is 15 minutes for all tasks and ad-hoc drivers. We use euclidian distances and assume a constant speed of 50km (31 miles) per hour. We ignore the service times at the picking and drop-off locations.

In the experiments, we assume that the costs of using an ad-hoc driver are proportional to the detour distance. For a backup vehicle we assume that the costs are proportional to the total route length. Moreover, we assume that the variable costs per distance unit are the same and equal to one for the backup vehicle and the ad-hoc driver. This implies that the costs are equal to the system-wide vehicle miles.

The characteristics of the base case instances are summarized in Table 3.

Table 3: Characteristics of base case instances

Definition	Values
No. of tasks	100
No. of ad-hoc drivers	100
Delivery lead-time (L)	90 min
Stop willingness of ad-hoc drivers (Q)	2
Announcement lead-time (A)	15 min
Departure time flexibility (F)	20 min
Vehicle speed	50 km/h

6.2 Benchmark

To evaluate the performance of our rolling horizon strategies, we report a *hindsight* benchmark solution that serves as a theoretical lower bound on the solution quality. Because it is not possible to determine all feasible jobs for the backup vehicles in reasonable time for the larger instances, we eliminate the fleet size restrictions for the backup vehicles in our matching formulation (see appendix 8.2). This allows us to reduce the number of jobs to consider, since we can decompose all large jobs in smaller individual dispatches (subsets of the complete route). The model then aims to minimize the costs of all backup dispatches plus the costs of using the ad-hoc drivers. As a post-processing step we can then determine the minimum number of vehicles required for this solution, as is shown in appendix 8.3.

For the benchmark solutions, we also implement the following heuristic speed-ups: (1) we keep track of at most five best routes for each job in the recursive algorithm, and (2) we limit the maximum route duration of the backup vehicles to 60 minutes. Table 4 shows that these speeds-up have a very small deteriorating effect on the solutions with a maximum optimality gap of less than 1% over all instance types.

Table 4: Heuristic validation for different instance types, L=90, Q=2

	% Δ OPT		
Instance	Avg. cost	Max. cost	# OPT
g1	0.00	0.00	10/10
g2	0.12	0.52	5/10
g3	0.02	0.17	9/10

6.3 Results

We evaluate the solutions in the various experiments by the following statistics:

- *Total cost*: the sum of the compensations of the matched drivers and the cost of the back-up trips.
- *Task-matched*: the fraction of tasks that are served by an ad-hoc driver; the complement represents the percentage of tasks that are served by the back-up services.
- *Driver-matched*: the number of ad-hoc drivers that are assigned to a job; some of the ad-hoc drivers that offered their service will not be used.
- *Back-up vehicles*: the number of backup vehicles that is required to serve all tasks.

6.3.1 Base analysis

For the base analysis, we present the results for both the dynamic setting and the hindsight benchmark. In Table 5, we compare the solutions for the three instance types (g1,g2,g3) for a stop willingness Q of 2 and 4: AHDR-2 and AHDR-4. As an additional benchmark, we also present the solution that we would obtain without ad-hoc drivers (DEDR), in which all

tasks are served by the dedicated backup vehicles. The cost of the DEDR solution is chosen as a baseline for the cost benchmark and its cost is normalized to 100.

Our method is fast and can solve the various offline problems to optimality within our rolling horizon framework in less than 15 seconds. Recall that we only have to find new jobs for the task or driver that triggered the new optimization run and reuse all jobs from the previous run that are still feasible.

From the results, we see that there are clear benefits from the use of ad-hoc drivers. Table 5 shows a reduction in costs from the use of ad-hoc drivers of between 18.8 and 37 percent as compared to the DEDR solution in the dynamic setting. We see the highest number of task matches and associated cost savings in the first geography. This is intuitive because in the first geography the drivers do not have to make detours to accommodate a pickup, since drivers and tasks all start at the same location. On the other end of the spectrum, the third geography has the lowest matching rate and savings. Figure 3 shows the absolute distances for the different instance types.

Recall that in our experiments the transportation costs are proportional to the system-wide vehicle miles. This implies that the cost reductions correspond to distance reductions. Note that the reduction of the total system-wide vehicle miles from the use of ad-hoc drivers may provide environmental benefits such as reduced emissions and congestion. This, however, is only the case if the ad-hoc drivers produce less or equal emissions per mile compared to dedicated drivers.

We also observe that the cost-efficiency of the system increases with the stop willingness of the drivers. This again relates to an increase in the number of tasks that are served by the ad-hoc drivers. The number of tasks matched increases for all instance types if the drivers' stop willingness increases from two to four. Interestingly, we see that in g1 and g3 we need fewer drivers when each individual driver can make more stops. This suggests that by combining the delivery of multiple tasks, we need fewer drivers to do the same amount of work.

If we compare our rolling horizon solutions to the (theoretical) hindsight benchmark, we

see a gap between 6 and 30 percent. We see that this gap is larger for the settings that include ad-hoc drivers than for the settings that do not. This makes sense as the settings with ad-hoc drivers involve more uncertainty and can therefore benefit more when all information is available. That is, while the dedicated backup vehicles are employed by a company, the ad-hoc drivers are private independent entities whose arrivals over time are difficult to predict. This also explains why more tasks are served by the ad-hoc drivers in the hindsight solutions than in the rolling horizon solution.

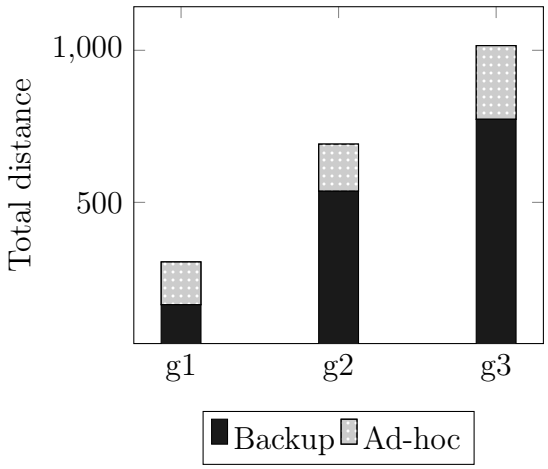
Table 5: Base analysis, base instances, 100 tasks, n=5

	Dynamic				Hindsight			
	Costs	% tasks matched	No. drivers matched	No. backup vehicles	Costs	% tasks matched	No. drivers matched	No. backup vehicles
g1								
AHDR-2	67.9	66.2	43.4	2.8	54.5	77	47.5	1.0
AHDR-4	63.0	77.0	40.2	2.6	50.6	84.4	51.2	1.0
DEDR	100	0.0	0.0	3.4	94	0.0	0.0	2.0
g2								
AHDR-2	77.5	42.2	36.8	4.8	62.2	49.2	40.4	2.4
AHDR-4	73.9	54.4	37.4	4.8	69.0	58.6	38	2.2
DEDR	100	0.0	0.0	6.4	90.1	0.0	0.0	4.6
g3								
AHDR-2	81.2	41.8	41.8	7.0	71.4	50.6	50.6	5.2
AHDR-4	78.8	53.6	40	6.8	66.7	64.2	43.4	5.0
DEDR	100	0.0	0.0	8.2	94.2	0.0	0.0	6.4

6.3.2 Comparison of commitment strategies

Up to now, we used a strategy that commits to tentative matches as late as possible. In this section, we study the impact of the so-called *earliest-commit* strategy, in which we commit to a tentative match as soon as we find it. While this strategy minimizes the waiting time for the drivers and the lead-time for the tasks, it may not be the best strategy in terms of the total system performance. Moreover, we also consider two hybrid strategies in which we commit the matches that involve an ad-hoc drivers early and commit a backup vehicle late

Figure 3: Total distances driven by the backup vehicles and the ad-hoc drivers, base instances, n=5



and vice versa. We normalize the costs of the default latest commit strategy to 100.

As expected, the results in Table 6 show that the latest commitment strategy outperforms all other strategies in terms of the total costs and number of tasks matched. Intuitively, we see that committing early to tentative matches that involve the backup vehicles has a significant negative impact on the results. An early committing to matches with the ad-hoc vehicles is less problematic since their limited availability provides less room to delay any commitments.

Table 6: Comparison of commitment strategies, base instances, g2, n=5

Commit ad-hoc, backup	Costs	% tasks matched	No. drivers matched	No. backup vehicles
Late, Late	100.0	42.2	36.8	4.8
Early, Late	101.2	37.8	34.4	5.2
Late, Early	121.6	28.8	27.2	4.8
Early, Early	122.6	26.2	25.2	5.2

6.3.3 Impact of departure time flexibility and delivery lead time

In this section, we examine the effect of the drivers' departure time flexibility and the delivery lead-times on the performance of the system. Table 7 presents the results for a departure

time flexibility of 10, 20, or 30 minutes, and a delivery lead time of 60, 90, or 120 minutes. Similar to the previous results, the costs are normalized to 100 for the base case. As expected, the costs decrease for higher departure time flexibilities and delivery lead-times. In a similar vein, the number of matched tasks increases with the departure time flexibility and with the delivery lead time.

Table 7: The impact of delivery lead-time and departure time flexibility, 100 tasks, base instances, g2, n=5

	Costs			% tasks matched		
Departure Time Flexibility →	10	20	30	10	20	30
Delivery Lead Time ↓						
$L = 60$	147.6	134	120.3	20	33.5	46.9
$L = 90$	112.9	100	86.3	28.9	42.2	56.7
$L = 120$	97.9	97.8	81.9	31.6	47.7	61.1

7 Concluding remarks

In this study, we introduce a variant of the dynamic pickup and delivery problem that aims to utilize the excess capacity of the existing traffic flow in urban areas. We consider a fleet of dedicated backup vehicles and a set of dynamically arriving ad-hoc drivers who are willing to make a small detour in exchange for a small compensation. We formulate the associated problem as a matching problem with side constraints. To handle real-time updates, we propose a rolling horizon framework that re-optimizes the system whenever new information becomes available.

We also investigate the viability of the crowdsourced delivery concept under the setting of a peer to peer platform. We test the performance of the platform with a simulation study based on three different instance types: a single origin, multiple origins and random origins. As expected, the time flexibility and the stop willingness of ad-hoc drivers have a strong impact on the performance of the system. Also, we compare the performance of the crowdsourced delivery system with a delivery system where all tasks are served by dedicated

drivers. The results indicate that there are clear benefits of using ad-hoc drivers in addition to a fleet of dedicated backup vehicles.

In this study, we assume that we do not have any probabilistic information on the arrival of delivery tasks and ad-hoc drivers. Thus, a stochastic version of this problem in which this kind of information is also taken into account looks like a natural direction for further research.

References

Agatz, Niels, Alan Erera, Martin Savelsbergh, Xing Wang. 2012. Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research* **223**(2) 295–303.

Agatz, Niels AH, Alan L Erera, Martin WP Savelsbergh, Xing Wang. 2011. Dynamic ride-sharing: A simulation study in metro Atlanta. *Transportation Research Part B: Methodological* **45**(9) 1450–1464.

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

Baldacci, Roberto, Vittorio Maniezzo, Aristide Mingozzi. 2004. An exact method for the car pooling problem based on lagrangean column generation. *Operations Research* **52**(3) 422–439.

Berbeglia, Gerardo, Jean-François Cordeau, Irina Gribkovskaia, Gilbert Laporte. 2007. Static pickup and delivery problems: a classification scheme and survey. *Top* **15**(1) 1–31.

Berbeglia, Gerardo, Jean-François Cordeau, Gilbert Laporte. 2010. Dynamic pickup and delivery problems. *European journal of operational research* **202**(1) 8–15.

Doan, Anhai, Raghu Ramakrishnan, Alon Y Halevy. 2011. Crowdsourcing systems on the world-wide web. *Communications of the ACM* **54**(4) 86–96.

Einav, Liran, Chiara Farronato, Jonathan Levin. 2015. Peer-to-peer markets. Tech. rep., National Bureau of Economic Research.

Fatnassi, Ezzeddine, Jouhaina Chaouachi, Walid Klibi. 2015. Planning and operating a shared goods and passengers on-demand rapid transit system for sustainable city-logistics. *Transportation Research Part B: Methodological* **81** 440–460.

Furuhata, Masabumi, Maged Dessouky, Fernando Ordóñez, Marc-Etienne Brunet, Xiaoqing Wang, Sven Koenig. 2013. Ridesharing: The state-of-the-art and future directions. *Transportation Research Part B: Methodological* **57** 28–46.

Ghilas, Veaceslav, Emrah Demir, Tom Van Woensel. 2013. Integrating passenger and freight transportation: Model formulation and insights. *Working paper, Technical University of Eindhoven* .

Klapp, Mathias A, Alan L Erera, Alejandro Toriello. 2016. The one-dimensional dynamic dispatch waves problem. *Transportation Science* .

Lee, Alan, Martin Savelsbergh. 2015. Dynamic ridesharing: Is there a role for dedicated drivers? *Transportation Research Part B: Methodological* **81, Part 2** 483 – 497. Optimization of Urban Transportation Service Networks Optimization of Urban Transportation Service Networks.

Li, Baoxiang, Dmitry Krushinsky, Hajo A Reijers, Tom Van Woensel. 2014. The share-a-ride problem: People and parcels sharing taxis. *European Journal of Operational Research* **238**(1) 31–40.

Li, Baoxiang, Dmitry Krushinsky, Tom Van Woensel, Hajo A Reijers. 2016. An adaptive large neighborhood search heuristic for the share-a-ride problem. *Computers & Operations Research* **66** 170–180.

Masson, Renaud, Anna Trentini, Fabien Lehuédé, Nicolas Malhéné, Olivier Péton, Houda Tlahig. 2014. Optimization of a city logistics transportation system with mixed passengers and goods. *EURO Journal on Transportation and Logistics* 1–29.

Mingozzi, Aristide, Lucio Bianco, Salvatore Ricciardelli. 1997. Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints. *Operations Research* **45**(3) 365–377.

Morphy, Erika. 2014. About Walmart's idea to crowdsource its same-day delivery service.

Popper, Ben. 2015. Drones could make amazon?s dream of free delivery profitable. URL `http://www.theverge.com/2015/6/3/8719659/amazon-prime-air-drone-delivery-profit-free-shipping-small-items`.

Rougès, Jean-François, Benoit Montreuil. 2014. Crowdsourcing delivery: New interconnected business models to reinvent delivery. *1 st International Physical Internet Conference*. 28–30.

Sadilek, Adam, John Krumm, Eric Horvitz. 2013. Crowdphysics: Planned and opportunistic crowdsourcing for physical tasks. *SEA* **21**(10,424) 125–620.

Savelsbergh, Martin, Tom Van Woensel. 2016. 50th anniversary invited article city logistics: Challenges and opportunities. *Transportation Science* **50**(2) 579–590.

Savelsbergh, Martin WP. 1985. Local search in routing problems with time windows. *Annals of Operations research* **4**(1) 285–305.

Stiglic, Mitja, Niels Agatz, Martin Savelsbergh, Mirko Gradisar. 2015. The benefits of meeting points in ride-sharing systems. *Transportation Research Part B: Methodological* **82** 36 – 53.

Suh, Kyo, Timothy Smith, Michelle Linhoff. 2012. Leveraging socially networked mobile ict platforms for the last-mile delivery problem. *Environmental science & technology* **46**(17) 9481–9490.

Voccia, Stacy A., Ann Melissa Campbell, Barett W. Thomas. 2015. The same-day delivery problem for online purchaces. *Working paper, University of Iowa* .

8 Appendix

8.1 MIP formulation TSP-TWPC

We can formulate the TSP-TWPC for a single driver k as a mixed integer problem. Recall that o_p, o_d and d_p, d_k represent origin and destination nodes for task p and driver k, respectively. The route always starts from the origin of the driver and ends at his destination. Let N^P be a set of nodes that correspond to the origins and destinations of the tasks in P, and let N be set of nodes including the origin and the destination of the driver. Let N^{P^+} and N^{P^-} denote the nodes associates with the origins and the destinations, respectively.

Let x_{ij} be a binary decision variable that is equal to 1 if the driver uses arc $(i, j), i, j \in N$ and 0 otherwise. Let c_{ij} be the cost of using arc (i, j). The continuous variable $B_i, i \in N$ represents the arrival time of the driver to node i. Then, the mixed integer problem can be

formulated as follows:

$$\min \sum_{i,j \in N} c_{ij} x_{ij} - c_{o_k,d_k} \tag{5}$$

subject to

$$\sum_{j \in N} x_{i,j} = 1 \qquad \forall i \in N^P, \tag{6}$$

$$\sum_{j \in N^p} x_{o_k,j} = 1 \tag{7}$$

$$\sum_{i \in N^p} x_{i,d_k} = 1 \tag{8}$$

$$\sum_{j \in N} x_{ij} - \sum_{j \in N} x_{ji} = 0 \qquad \forall i \in N^p, \tag{9}$$

$$\sum_{i,j \in N} I_{pl(i) \neq pl(j)} x_{ij} \leq Q_k \tag{10}$$

$$B_{d_p} \geq t_{o_p,d_p} + B_{o_p} \qquad \forall p \in P, \tag{11}$$

$$B_j \geq B_i + t_{ij} - M(1 - x_{ij}) \qquad \forall i, j \in N, \tag{12}$$

$$e_i \leq B_i \leq l_i \qquad \forall i \in N, \tag{13}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in N$$

$$B_i \geq 0 \qquad \forall i \in N,$$

The objective (5) is to minimize the total travel costs to serve all delivery tasks by the driver. Constraint (6) ensures that each task is served exactly once. Constraints (7) and (8) make sure that the driver starts at his origin and ends at his destination. Equations (9) represent the flow conservation constraints. Constraint (10) ensures the maximum number of stops per driver. In constraint (10), the indicator function $I : L^2 \mapsto 0, 1$ controls the number of different physical locations that the driver visits in his tour. Constraints (11) ensure the precedence relations between pickup and delivery points. Constraints (12) and (13) represent the time window constraints.

8.2 Matching formulation hindsight benchmark

Note that a dispatch node is not associated with a particular backup vehicle but only denotes a possible dispatch at a certain time from a certain location.

Let A be the set of all feasible arcs. Let $J_k,\ \ k \in D$ denote the collection of jobs that driver k can serve, and $J_p,\ \ p \in P$ denote the set of jobs that contains task p. Let x_{kj} be the binary decision variable that indicates whether the arc between driver k and job j is in the solution ($x_{kj} = 1$) or not ($x_{kj} = 0$) and let y_j be the binary decision variable that indicates whether job j is served by a backup vehicle ($y_j = 1$) or not ($y_j = 0$). The coefficient c_{kj} represents the weight of the arc (k, j), which denotes the cost if driver k is assigned to job j (where c_j is the cost when job j is served by a backup vehicle.) Then, the problem that aims to minimize the total cost can be formulated as follows:

$$\min \sum_{(k,j)\in A} c_{kj}x_{kj} + \sum_{j\in J} c_j y_j \tag{14}$$

$$\text{s.t} \sum_{j\in J_k} x_{kj} \leq 1 \ \ \forall k \in D, \tag{15}$$

$$\sum_{j\in J_p} \sum_{k\in D} (x_{kj} + y_j) = 1 \ \ \forall p \in P, \tag{16}$$

$$x_{kj} \in \{0, 1\} \ \ \forall (k, j) \in A.$$

Equation (14) is the objective function that aims to minimize the sum of the costs of the matches and the backup vehicles. Constraints (15) makes sure that each driver is assigned to at most one job. Note that this constraint only applies to the ad-hoc drivers since we do not explicitly restrict the number of backup vehicles in this formulation. Constraints (16) make sure that each job is assigned to one of the drivers or a backup vehicle.

8.3 Determining the number of backup vehicles

The solution to the above problem formulation provides the matches between the jobs

and the ad-hoc drivers and the selected backup dispatches. The subset of backup dispatches that is selected in the solution is denoted by $B' \subseteq B$. To determine the minimum number of required backup vehicles to cover all dispatches, we need to solve a problem that resembles an interval scheduling problem.

We model this problem on a directed graph $G = (V, A)$, where $V = \{s, B', t\}$ includes a node for each dispatch $b \in B'$, a source node s and sink node t. Let A be the set of arcs. We create an arc from s to every dispatch in B' and from every dispatch in B' to t and an arc between every pair of dispatches that could potentially be served by the same vehicle. In particular, we only add an arc (i, j) between dispatch i and j if: $l_j \geq e_i + \tau_i$, where $[e_i, l_i]$ is the dispatch time window, and τ_i is the duration of dispatch i. Let x_{ij} be a binary variable that is equal to 1 if arc $(i, j) \in A$ is selected, and 0 otherwise. Let t_b^d be the actual dispatch time of dispatch b. This gives the following formulation:

$$\min \sum_{i \in B'} x_{si} \tag{17}$$

$$\text{s.t} \sum_{i \in V \setminus \{t\}} x_{ij} = 1, \quad \forall j \in B' \tag{18}$$

$$\sum_{i \in V \setminus \{t\}} x_{ib} - \sum_{j \in V \setminus \{s\}} x_{bj} = 0, \quad \forall b \in B' \tag{19}$$

$$t_j^d \geq (t_i^d + \tau_i) x_{ij}, \quad \forall i \in B', j \in B' \tag{20}$$

$$e_i \leq t_i^d \leq l_i, \quad \forall i \in B' \tag{21}$$

$$x_{ij} \in \{0, 1\}, \quad (i, j) \in A$$

The objective function (17) minimizes the outflow from the source node, which is equivalent to minimizing the number of backup vehicles. Constraint (18) ensures that each dispatch is covered by exactly one vehicle. Constraint (19) guarantees that the inflow and outflow of each node are consistent. Constraints (20) and (21) ensure that the dispatch time windows are satisfied.

8.4 Proof of Observation 3

This proof is constructive and provides a procedure to transform an arbitrary route into a *clustered* route.

We start with an arbitrary route r which is fully characterised by the ordered sequence of pick-ups and deliveries. The pick-up of task j reads p_j, and the delivery of task j reads d_j. We write that the route will start with k pick-ups, where $k \geq 1$, and then l deliveries will follow, where $l \leq k$, before another pick-up (of task $k+1$) will follow. The sequence of the first $k+l+1$ pick-ups and deliveries can be written as

$$p_1, \ldots, p_k, d_{j_1}, \ldots, d_{j_l}, p_{k+1}.$$

Here the deliveries correspond with a subset of the tasks of the pick-ups, i.e., $\{j_1, \ldots, j_l\} \subseteq \{1, \ldots, k\}$; an item can only be delivered once it has been picked up.

In the case when $k = l$, the first k pick-ups have all been delivered, so the vehicle returns to the next pick-up p_{k+1} empty, and it is also possible that the job has been completed, so that there is no next pick-up $k+1$. In that case, we are finished. In the case that there is another pick-up remaining, we can proceed with the analysis as above with the first pick-up p_{k+1}.

In the case when $k > l$, the first k pick-ups have not yet all been delivered, so the vehicle is not empty when it returns to pick-up task $k+1$. We need to expand the sequence of pick-ups and deliveries further. The route r can be written as

$$p_1, \ldots, p_k, d_{j_1}, \ldots, d_{j_l}, p_{k+1}, \ldots, p_{k+r}, d_{j_{l+1}}, \ldots, d_{j_{l+s}}, p_{k+r+1}, \ldots,$$

where $r \geq 1$ and $l + s \leq k + r$.

Consider p_i, with $1 \leq i \leq k$. In case $i \notin \{j_1, \ldots, j_l\}$, task i is carried back to the pick-up location of tasks $k+1, \ldots, k+r$. We need to consider the following two cases:

1. The pick-up location of task i is not revisited while picking up tasks $k+1, \ldots, k+r$.

This implies that this task does not violate the clustered route property during the first $k + r$ pick-ups.

2. The pick-up location of task i is revisited while picking up tasks $k + 1, \ldots, k + r$; let task $j \in \{k+1, \ldots, k+r\}$ be the first task in the sequence which has the same pick-up location as i. Then we change the order of pick-ups and deliveries in the route by moving the pick-up of task i right in front of the pick-up of task j. The total distance travelled en route will not increase, and the pick-up of task i will simply be delayed until the moment that task j will be picked up at the same location. The pick-up of task i still occurs before task i is delivered.

This procedure can be followed for all tasks $i \notin \{j_1, \ldots, j_l\}$. The order in which these tasks are considered may have an impact on the order in which tasks are picked up at the same location, but this is not relevant.

We have now arrived at a sequence of pick-ups and deliveries where the first subsequence of pick-ups $\{p_i\}$ and the first subsequence of deliveries $\{d_j\}$ satisfy the clustered route property. We now need to proceed to analyse the remaining sequence of pick-ups and deliveries in a similar manner. Note that some of the tasks, which have been picked up in the first subsequence, may not have been delivered yet in the first subsequence of deliveries. They should be incorporated in the further analysis as well, but this does not change the line of argument.

The transformation of the route by pushing forward deliveries as described above is completed after a finite number of steps. The resulting route has the clustered route property. □