# Crowdsourced Delivery - a 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

February 2, 2016

**Abstract**

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 matching of tasks and drivers 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 environmentally friendly.

# 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, E. 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 (Instacart, Postmates, Deliv, Trunkrs and Hitch). 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. The drivers pick up parcels from a retail store, warehouse or dedicated pickup location 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

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 complex.

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. This setting is highly dynamic with both tasks and drivers continuously arriving over time. This means that the crowdsourcing provider needs to assign delivery tasks to drivers and to determine the sequence of the associated stops.

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 to perform on-demand deliveries. We present a rolling horizon framework and develop an exact solution approach

to solve the different subproblems. 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 environmental benefits, we compare the performance of a crowdsourced system with a traditional dedicated delivery system.

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 sections 4 and 5 we formulate the off-line problem and provide a solution approach. In section 6 we explain the implementation of our rolling horizon framework. In section 7, we describe our instances and present the results from our numerical experiments. Finally, in section 8 we provide some concluding remarks and directions for future research.

## 2 Related literature

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)). A particular dynamic problem variant that has recently started to receive some attention involves the same-day delivery of goods ordered online from a single depot (see Klapp et al. (2015) and Voccia et al. (2015)).

Unlike the traditional PDP setting, in our problem we do not have a dedicated fleet of vehicles and employees available to serve all jobs. In that sense, our problem is similar to ride-sharing 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 passenger, 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. (2015) 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 neighbourhood 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 $l, m \in N$.

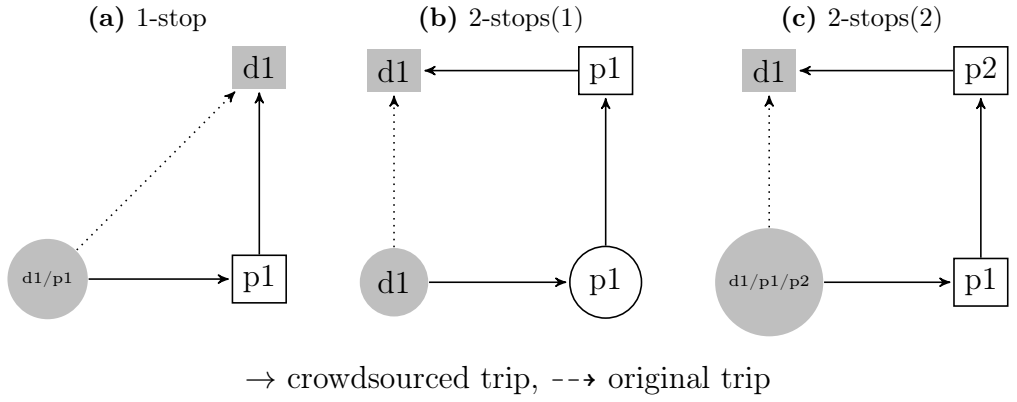
Let $P$ be the set of delivery task (parcel) announcements. Delivery task $p \in P$ has a pickup location $o_p$, which can be a retail store, warehouse or dedicated pickup point, and a drop-off location $d_p$, which is usually the home of the online buyer. The task has an earliest pickup time $e_p$ when 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 delivery 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 $D$ be the set of driver announcements. The driver's trip announcement $k \in D$ specifies his origin $o_k$ and destination $d_k$. A driver $k \in D$ 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 $Q_k$ 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.

Serving a task is associated with at most two stops: one at the pickup location and one at

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




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 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 Fung, Brian (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 are in at least one feasible match. A match $(k, j, r)$ 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

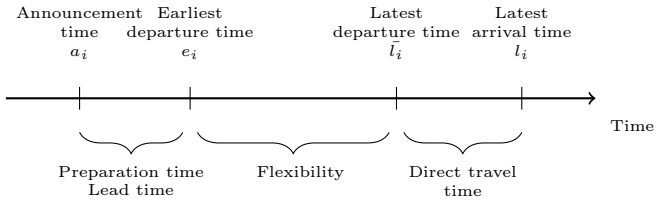
equal to $Q_k + 2$ (including the origin and destination of the driver).

- *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.

The system is characterized by the continuous arrival of drivers and tasks. Each driver $k \in D$ 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 to 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. When it is not possible to serve a certain task before its deadline $\bar{l}$, the delivery platform uses an emergency backup option to serve the task. The third party backup option is more expensive than using the ad-hoc drivers: the per-mile and fixed cost

of the back-up option is $\alpha \geq 1$ times the costs of an ad-hoc driver since this service requires excessive availability towards to the real-time requests without rejecting any of them.

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 emergency back-up costs. In this model, the platform aims to maximize the total profit. Since the platform accepts all delivery tasks, the objective of profit maximization is equivalent to minimizing total system-wide delivery cost.

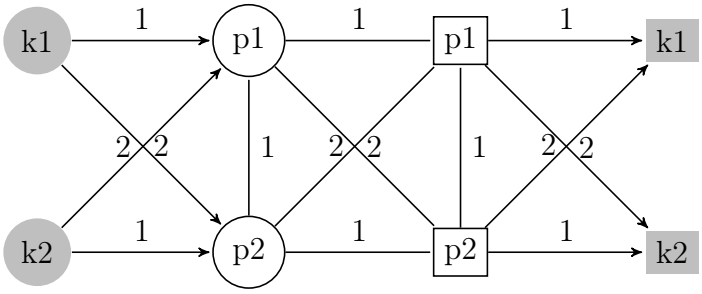
# 4 Offline problem formulation

As in Stiglic et al. (2015), we can model this problem as a matching problem with side-constraints. We create a node for each driver $k \in D$ and a node for each job $j \in J$. An arc between node $k$ and node $j$ represents a feasible match between driver $k$ and job $j$. The weight of the arc denotes the savings of serving job $j$ by driver $k$ as compared to serving it by a back-up vehicle. Note that when a job $j$ contains only a single task, there exists only one route, which is the origin of the driver and the task follow 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.

An example of the subproblem is depicted in Figure 3 with two drivers $k1$ and $k2$ and two tasks $p1$ and $p2$, where the numbers above the arcs represent the time amount of travel between the nodes. In this example, both drivers are willing to accept an increase in their original trips at most ten minutes. It is obvious that any route option is feasible for both drivers; however, the shortest route for $k1$ that serves both tasks is $o(p1), o(p2), d(p2), d(p1)$ while the shortest route for $k2$ is equal to $o(p2), o(p1), d(p1), d(p2)$. Additionally when time windows are included into the example, the subproblem becomes a well known travelling salesman problem with time windows and precedence constraints (TSP-TWPC). Here, the precedence constraints ensure that drivers pick up tasks before dropping them off. Different from the traditional TSP-TWPC is that we also restrict the stop numbers of each driver with

their stated willingness. The mathematical formulation of this subproblem can be found in Appendix A.

**Figure 3:** Two drivers (grey) and two tasks (white) traveling from origin (circle) to destination (square)




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 ($x_{kj} = 0$). The coefficient $s_{kj}$ represents the weight of the arc $(k, j)$, which denotes the cost savings if driver $k$ is assigned the job $j$ as compared to the backup service. Then, the problem that aims to maximize the total cost savings can be formulated as follows:

$$\max \sum_{(k,j)\in A} s_{kj} x_{kj} \tag{1}$$

$$\text{s.t} \sum_{j\in J_k} x_{kj} \leq 1 \ \ \forall k \in D, \tag{2}$$

$$\sum_{j\in J_p} \sum_{k\in D} x_{kj} \leq 1 \ \ \forall p \in P, \tag{3}$$

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

Equation (1) is the objective function that aims to maximize the sum of the savings of the matches over the backup option. Constraints (2) and (3) make sure that each driver is assigned to at most one job and each job is assigned to at most one driver.

# 5 Solution method

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) since it involves solving the TSP-TWPC as a subproblem. Savelsbergh (1985) showed that finding a feasible solution of TSP-TW is NP-complete. However, in our problem setting, the number of tasks per job is relatively small; thus, we can solve each problem very fast.

Next, we describe how to determine all feasible jobs to be included in the matching problem. Note that we have to determine not only the feasibility of serving a specific set of tasks but also the sequence in which to serve them. In the worst case, the number of feasible matches for $p$ tasks and $k$ drivers is $O(k2^p)$ (if each driver can serve all tasks in one trip). However, due to the time and stop restrictions, the number of feasible routes is likely to be far less in practice. The following two observations are the backbone of our recursive algorithm.

**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, see 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 in a recursive algorithm to reduce the number of jobs to be searched by using a 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 adding 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.

Based on these two observations, all feasible jobs with respect to the driver $k \in D$ can be determined by using a recursive algorithm. Naturally, the recursion starts with determining the jobs with single tasks and combine 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 of $D$ drivers and $P$ tasks.
Initialize: Find all feasible pairs of a driver and a single-task job. Store all feasible single-task jobs for driver $k$ in $J_k^1$ and construct the route set $R_{k,j} = \{r_{k,p} | p \in j\}$ for each pair of $(k, j)$.

**for all** $k \in D$ **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 and single-task job, we create a route set $R_{k,j}$, which contains all feasible routes that driver $k$ can make to serve job $j$. Note that for single-size jobs, $R_{k,j}$ includes only a single route. 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 into the set $R_k$.

Next, Algorithm 1 repeatedly generates feasible jobs and routes for each driver. For a chosen driver $k$, the algorithm checks tuples with $w$ tasks by combining a feasible job $j$ whose size is $w-1$ with a single-task job $j' = \{p\}$ in $J_k^1$. For each tuple, the algorithm calls the subroutine `SUBFEAS`, presented in Algorithm 2, to check whether the tuple is feasible or not. `SUBFEAS` is nothing other than the implication of **Observation 1**. If `SUBFEAS` returns a yes answer, which means that each subset of $j \cup j'$ containing task $p$ is feasible, then the algorithm calls the subroutine `FINDROUTES`, given in Algorithm 3. `FINDROUTES` returns all feasible routes by inserting the origin and the destination of the task $p$ into all feasible routes of job $j$. If `FINDROUTES` returns a non-empty set, this means the tuple $j \cup j'$ is a feasible job for driver $k$. Then, the algorithm adds the new feasible job $j \cup j'$ into the set $J_k^w$. Also, the algorithm stores all associated feasible routes in the set $R_k$. Recursive algorithm terminates for each driver $k$, either when the size of the tuples being searched reaches the stop willingness of driver $k$ or when the set $J_k^w$ is empty. In the second case, obviously it is impossible to find a job with a size greater than $w$.

**Algorithm 2** Check feasibility of all subsets of the combination of job $j$ and task $p$ for driver $k$.

```
1: function SUBFEAS(j, p, k)
2:     z ← true
3:     for all j ∈ J_k^|j| do
4:         j' ← {(j \ {q}) ∪ {p}},  ∀q ∈ j
5:         if j' ∉ J_k^|j| then
6:             z ← false, return z
7:         end if
8:     end for
9:     return z
10: end function
```

**Algorithm 3** Create all feasible routes of the combination of job $j$ and task $p$ for driver $k$

```
1: function FINDROUTES(R_{k,j}, p)
2:     z ← ∅
3:     p_1 ← o_p
4:     p_2 ← d_p
5:     for all r_{k,j} ∈ R_{k,j} do
6:         for i ← 0 to |r_{k,j}| − 1 do
7:             if feasible(i, p_1) then
8:                 for l ← i + 1 to |r_{k,j}| do
9:                     if feasible(l, p_2) then
10:                        z ← z ∪ [r_{j,k} ∪ (p_1, p_2)]
11:                    end if
12:                end for
13:            end if
14:        end for
15:    end for
16:    return z
17: end function
```

# 6 Rolling horizon framework

Since both delivery tasks and drivers arrive dynamically throughout the day, we use an event-based rolling horizon framework that repeatedly solves the off-line problem each time $t$ that a new task or driver arrives. At each iteration $q$ of the rolling horizon approach, we solve the off-line problem for all active (known and still available) tasks and drivers. At time $t$, task $p$ is potentially active if it 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 $t$.

In the so-called *earliest-commit* strategy, 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. That is, we may be able to get better matches (i.e., less detour, less stops, more tasks) if we wait until the latest time to commit to a match. In the *latest-commit* strategy, we do not commit to a tentative match before its latest departure time. The latest departure time of a certain tentative match $(k, j, r)$ 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.

For a new optimization run $q$, we only have to create new jobs for the task or driver that arrived after 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. Jobs that are still feasible can be added to the matching problem without further pre-processing. Job that are not time feasible need to be re-evaluated. Since creating the feasible job variables is the most time consuming-step in our optimization approach we can solve larger sizes within the dynamic setting.

# 7 Computational experiments

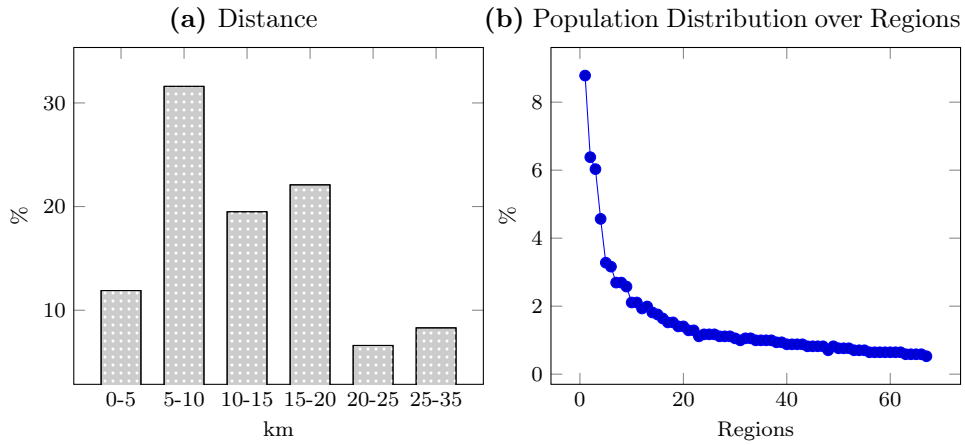
In this section, we discuss the performance of our exact solution approach and of the crowdsourced delivery platform in different settings. Section 7.1 describes the instances that we used in the various experiments and section 7.2 presents our results. In section 7.3, we compare the performance of a crowdsourced delivery system with a traditional system without ad-hoc drivers.

## 7.1 Instance generation

To evaluate the viability of the crowdsourced delivery platform, and to test the performance of the system, we generate several artificial instances based on data from a large Dutch multi-channel retailer. For February and March 2015, we have data on the address locations of the customers that made a purchase at their store location in the city of Rotterdam in the Netherlands. We generate our instances based on the customer density from the data by clustering the addresses of the customers by their 4-digit zip codes (each zip code is less than a square-km in size). Figure 4a and Figure 4b provide information about the density and distance statistics.

We generate the delivery tasks as follows. The origin of each delivery task is one of five potential locations with equal likelihood, one of which is the store location of our retailer. The destination of each delivery task is one of the 67 zip codes based on the customer

**Figure 4:** Overview of the Rotterdam Instances




distribution, where the exact location is a random point within one kilometer radius around the center of the zip code. For the drivers, the origin is one of ten potential locations with equal likelihood, five of which are the same as the potential origins of the delivery tasks. This means that we assume that half of the drivers start their trip from one of the stores or pickup locations of the tasks. The driver destinations are generated in the same way as for the delivery tasks.

The distance between two locations is measured by the haversine formula with a 30% uplift to reflect the urban road network. We assume a constant vehicle speed of 30km/h.

All drivers and delivery tasks arrive in a specific service period $[0, \bar{T}]$. We assume that the stores/ pickup locations are open between time 1 and $\bar{T}$. The tasks that arrive between time 0 and 1 can be considered as the accumulated demand over the night. To make sure that we can potentially serve all delivery tasks, the platform stops accepting delivery tasks at $\bar{T} - \max_{p \in P}(L_p)$. In particular, the announcement times of the delivery tasks follows a uniform random variable with parameters 0 and $L$, where $L$ is the system-wide lead-time. Drivers' announcement times are drawn uniformly from the interval between 0 and $\bar{T}$.

In our experiments, we assume that each driver has the same *system-wide* departure time flexibility, stop willingness and announcement lead-time. All parcel delivery tasks have the same delivery lead-time. Table 1 provides the base case values of our parameters. In terms

**Table 1:** Base case parameters

| Parameter | Definition | Base Case Value |
|---|---|---|
| $n_p$ | Number of tasks | 100 |
| $n_k$ | Number of drivers | 100 |
| $\bar{T}$ | Service period | 10 hours |
| F | Departure flexibility | 20 minutes |
| Q | Stop willingness | 2 |
| L | Delivery Lead-time | 90 minutes |
| A | Announcement Lead-time | 15 minutes |

of costs, we assume that a driver receives \$4 for each task that he delivers and \$1 per mile of his detour. This pricing scheme is in line with the model of the crowdsourced delivery start-up Postmates. When a task is sent by an emergency back-up option, the delivery platform incurs 1.1 times higher unit cost than sending with an ad-hoc driver.

## 7.2 Results

All experiments were implemented in C++ and conducted on a Pentium(R) Dual-Core CPU T4500 2.30 GHz and 2GB of installed RAM. Gurobi 5.63 was used as an IP solver. The results of each class of instances are an average over ten controlled arrival streams.

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 fraction of drivers that are assigned to a job.
- *Additional stops per driver*: the average number of additional stops performed by the matched drivers. This corresponds to the number of different locations visited by the drivers.
- *Detour per driver*: the average detour distance of the matched drivers relative to their original trip distance.

- *Tasks per driver*: the average number of tasks assigned to a matched driver.
- *Additional stops per task*: the average number of stops per crowdsourced delivery task. Since the delivery of a task involves at most two stops this value can not be more than two.

### 7.2.1 Baseline results

For the baseline results, we present the results obtained by solving an off-line problem in which all tasks and drivers is known in advance. In Table 2, we compare the solution for the base case with two additional system densities: 50:50 and 200:200 drivers and tasks. We run all experiments for a stop willingness $Q$ of 1, 2 and 4. The cost of the base scenario is chosen as a baseline for the cost benchmark and its cost is normalized to 100.

As expected, the costs-efficiency of the system improves with the system density, i.e. the cost per delivery decreases. This improvement in efficiency is mainly due to a higher number of tasks that is served by the ad-hoc drivers. The percentage of *Task-matched* goes up from 69.6% in the 50:50 case to 85.5% in the 200:200 case. Moreover, the number of tasks per driver also increases with more density. We do not only find more matches with more tasks, but also the quality of the matches increases. That is, the average detour decrease with the density.

**Table 2:** Base results for different numbers of participants and maximum stops

| Tasks:drivers ($n_p : n_k$) | 50:50 | | | 100:100 | | | 200:200 | | |
|---|---|---|---|---|---|---|---|---|---|
| Stop willingness ($Q$) | 1 | 2 | 4 | 1 | 2 | 4 | 1 | 2 | 4 |
| Total costs | 122.6 | 100.0 | 94.7 | 128.6 | 100.0 | 93.0 | 134.5 | 100.0 | 90.6 |
| Tasks-matched (%) | 27.2 | 69.6 | 74.8 | 37.2 | 76.4 | 82.4 | 46.5 | 85.5 | 88.3 |
| Drivers-matched (%) | 27.2 | 67.8 | 52.8 | 37.2 | 68.4 | 50.6 | 46.5 | 69 | 48.2 |
| Add. stops per driver | 1.0 | 1.7 | 2.4 | 1.0 | 1.8 | 2.6 | 1.0 | 1.5 | 2.7 |
| Detour per driver (%) | 63.6 | 56.1 | 59.5 | 58.0 | 49.4 | 50.9 | 58.1 | 46.1 | 41.7 |
| Tasks per driver | 1.0 | 1.2 | 1.4 | 1.0 | 1.2 | 1.6 | 1.0 | 1.2 | 1.8 |
| Add. stops per task | 1.0 | 1.8 | 1.7 | 1.0 | 1.6 | 1.6 | 1.0 | 1.9 | 1.5 |

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. In the base case, the number of tasks matched increases from 37.2% to 82.4% if the drivers' stop willingness increases from one to four. In terms of the number of matched drivers, the impact is less clear. We see that more drivers are matched when the stop willingness increases from one to two. Interesting, however, the number decreases when the stop willingness increases from two to four. To explain this behaviour, we should look at the number of tasks per driver. That is, while the number of matched tasks increases with the stop willingness, the number of tasks per driver shows a stronger increase. If we can combine the delivery of multiple tasks, we need less drivers to do the same amount of work.

If we focus on the number of additional stops per driver and per task, we see that the number of additional stops per task decreases with the stop willingness of drivers. However, also the average additional stop number grows with the stop willingness. The rationale of the result is intuitive. If a driver is willing to stop more, the possibility of taking more parcels from the same origin is higher, which leads both higher number of additional stops and better consolidation.

**Figure 5:** Number of stops

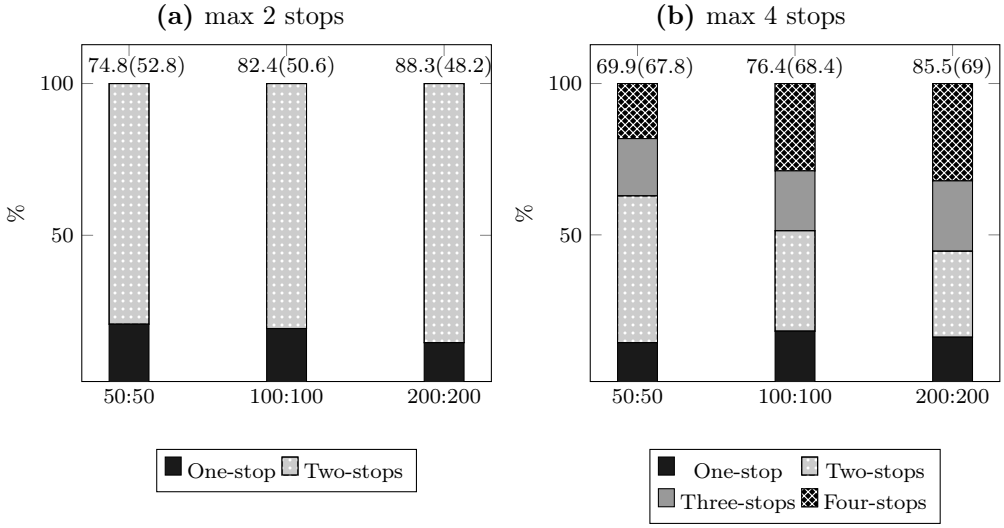



Figure 5 shows the fraction of drivers that make one to four additional stops for the different density scenarios. The numbers on the top of each bar represent the *Task-matched* and the *Driver-matched* values of the associated scenarios, respectively. As expected, more drivers

make more additional stops if the density increases. The main reason for this behaviour is that the denser settings increase the chances of combining tasks originated from the same location.

Table 3a presents the time that is required to create all nodes and arcs and Table 3b presents the time that is required to solve the matching problem. As expected, the computation times increase with the number of participants and the stop willingness. When the number of tasks and drivers are 50 and 50, the algorithm takes less than 20 seconds for all cases of the stop willingness. However, the required time to solve the problem with 200 tasks and 200 drivers is around 10 minutes.

**Table 3:** CPU time in sec.

**(a)** Pre-processing

| | 1 | 2 | 4 |
|---|---|---|---|
| 50:50 | 1.0 | 1.8 | 12.8 |
| 100:100 | 2.6 | 7.1 | 124.8 |
| 200:200 | 10.7 | 56.9 | 641.6 |

**(b)** Matching problem

| | 1 | 2 | 4 |
|---|---|---|---|
| 50:50 | 0.1 | 0.1 | 0.3 |
| 100:100 | 0.1 | 0.4 | 2.8 |
| 200:200 | 0.4 | 1.2 | 7.2 |

The results show that the majority of the computation effort is made to complete the pre-processing part. However, this effort contributes to solving the corresponding matching problems rapidly. For all scenarios, solving the matching problem takes less than 10 seconds. Although the time to solve matching problem also grows proportionally, for realistic-size instances the computation time can be easily neglected.

This result also justifies why our matching formulation is a practical approach in handling real-time responses. Recall from our rolling horizon approaches, presented in 6, we do not need to re-construct each feasible matches from the beginning. For each rolling iteration, our algorithm updates the set of feasible matches by simple check and add new set of feasible matches solely aroused by new arrivals. Thus, the corresponding pre-processing part of the problem is significantly smaller.

### 7.2.2 Impact of time flexibility and delivery lead-time

We examine the effect of the driver's time flexibility on the performance of the platform. We vary both the departure time flexibility and the detour flexibility, $T = t_{o_k,d_k} + F$. Table 4 shows that the performance of the system improves with the time flexibility of the drivers. We see that the benefits of additional time flexibility are bigger with the higher number of stop willingness. With a stop willingness of 4, we see that the cost of the system decreases with 27 percent points from 111.7 to 84.4. With a stop willingness of 2, the improvement is less pronounced. We see that marginal benefits decrease with the time flexibility. The main reason is that the time flexibility of the driver starts to become an unbinding constraint with respect to maximum stop willingness.

**Table 4:** Base results for different number of maximum stops and time flexibilities (100:100)

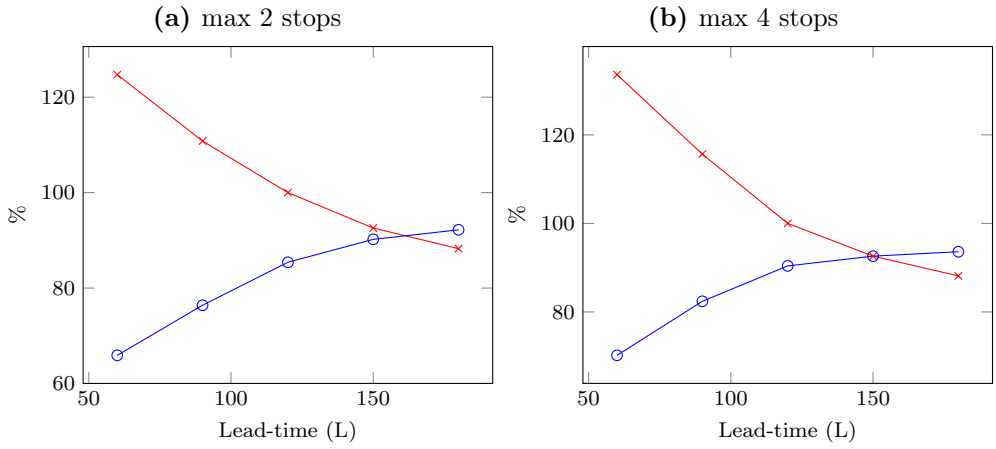
| Time flexibility ($F$) | 10 min. | | 20 min. | | 30 min. | |
|---|---|---|---|---|---|---|
| Stop willingness ($Q$) | 2 | 4 | 2 | 4 | 2 | 4 |
| Total costs | 114.6 | 111.7 | 100 | 93.0 | 94.9 | 84.4 |
| Tasks matched (%) | 55.6 | 58.2 | 76.4 | 82.4 | 83 | 91.2 |
| Drivers matched (%) | 51.4 | 44.8 | 68.4 | 50.6 | 71 | 52.4 |
| Add. stops per driver | 1.7 | 2.1 | 1.8 | 2.6 | 1.8 | 2.7 |
| Detour per driver (%) | 25.3 | 24.6 | 49.5 | 50.9 | 62.3 | 61.8 |
| Tasks per driver | 1.1 | 1.3 | 1.1 | 1.6 | 1.2 | 1.7 |
| Add. stops per task | 1.6 | 1.6 | 1.6 | 1.6 | 1.6 | 1.5 |

Next, we examine the impact of the delivery lead-time of the tasks $L$ on the performance. Figure 6 shows the results for five different delivery lead-times between 60 and 180 minutes for a stop willingness of 2 and 4. It is clear that larger lead-times yield lower total cost and higher match ratios.

### 7.2.3 Impact of dynamics

Table 5 shows the results of our rolling horizon implementation in which tasks and drivers arrive dynamically throughout the day for our two commitment strategies. For the reader's convenience, we also include the results for the off-line/ hindsight problem as a benchmark. In the real-time setting, we observe an increase in cost of $9-21\%$ as compared to the hindsight benchmark. As expected, we see that in the dynamic setting we serve less tasks per driver,

**Figure 6:** The impact of the delivery lead-time: $x$ = costs, $o$ = tasks matched




on average. We also see that for all instances, the *Latest-commit* strategies outperforms the *Early-commit* strategy by 6 to 9 percentage points. This illustrates that there is a clear benefit in waiting longer before committing to a match.

**Table 5:** Base results with rolling horizon for different commitment strategies (100:100)

| | offline. | | early | | late | |
|---|---|---|---|---|---|---|
| Stop willingness ($Q$) | 2 | 4 | 2 | 4 | 2 | 4 |
| Total costs | 100 | 93.0 | 115.6 | 113.5 | 109.5 | 103.9 |
| Tasks matched (%) | 76.4 | 82.4 | 71.4 | 77 | 69.8 | 77.4 |
| Drivers matched (%) | 68.4 | 50.6 | 68.2 | 60.2 | 65 | 50 |
| Add. stops per driver | 1.8 | 2.6 | 1.8 | 2.2 | 1.8 | 2.6 |
| Detour per driver (%) | 49.5 | 51.0 | 66.0 | 76.6 | 53.9 | 63.7 |
| Tasks per driver | 1.1 | 1.6 | 1.0 | 1.3 | 1.1 | 1.5 |
| Add. stops per task | 1.6 | 1.6 | 1.7 | 1.8 | 1.6 | 1.7 |

## 7.3 Dedicated drivers benchmark

To evaluate the performance of our crowdsourced delivery system over the traditional delivery methods, we propose a benchmark that represents a system where all deliveries are made by dedicated drivers. A dedicated driver is an employee and therefore does not have any stop and detour time restrictions. Additionally, the dedicated drivers are ready to serve parcels as long as they do not violate any delivery deadlines. Since the delivery service is made by only the dedicated vehicles in the benchmark, the method can also be called as traditional

delivery system. As similar to the formulation of many pickup and delivery problems, we choose one of the pick-up locations as a depot from where all dedicated drivers start and end. All other assumptions for the ad-hoc drivers hold also for dedicated drivers. In this benchmark, the solutions of the crowdsourced delivery system and the traditional delivery system method are obtained by solving a real-time problem in which the announcements are not known in advance.

Without the time and stop restrictions of the drivers, the number of subproblems that have to be solved in the pre-processing phase becomes too many and the time required to solve each one grows exponentially. Therefore, we introduce a heuristic procedure that is similar to the solution approach as described in Section 5. Let $D^d$ be the set of dedicated drivers and let $R_k$ be the set of feasible routes of the dedicated driver $k \in D^d$. When a delivery task $p$ arrives, we call the subroutine `FINDROUTES` described in the Algorithm 3 with inputs $R_k$ and $p$ for each driver that is at the depot at the time. Recall that the Algorithm 3 inserts the origin $o_p$ and the destination $d_p$ of task $p$ into the feasible routes of driver $k$, $R_k$. Then, the heuristic selects the best driver and the best route among all feasible routes found. For the selected driver $k'$, we update $R_{k'}$ with a new set of routes that is a return of the `FINDROUTES`$(R_{k',p})$. Note that corresponding to his best route, each driver $k \in D^d$ has a latest departure time. The latest departure time of a dedicated driver is the final time that he is able to serve all tasks in the route without violating any constraint. At the latest departure time, dedicated driver $k$ starts his route and returns the depot after finishing all deliveries. Note that the procedure proposed for the dedicated drivers to handle real time responses is the exact same method as *Latest-Commitment* introduced earlier in Section 6.

**Figure 7:** Additonal stops

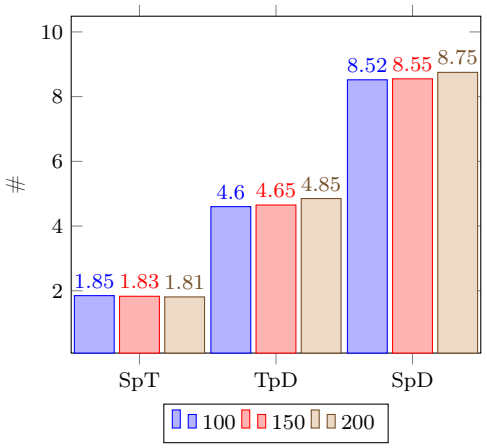



**Table 6:** A distance over the benchmark

| (Stop willingness $Q$) | 2 | 4 |
|---|---|---|
| 100:100 | 88.4% | 84.0% |
| 150:150 | 91.4% | 83.9% |
| 200:200 | 85.7% | 78.7% |
| 100:150 | 79.2% | 75.5% |
| 150:200 | 83.3% | 76.1% |
| 150:100 | 98.7% | 90.6% |
| 200:150 | 90.6% | 83.0% |

Figure 7 presents the information of the average number of stops per task (SpT), the average number of tasks per delivery tour (TpD), and the average number of stops per delivery tour (SpD) with varying numbers of total tasks. It can be seen that while the average number of stops per task decreases in total number of tasks, the average number of tasks per delivery tour increases. As expected, these trends reveal that the traditional delivery systems also yield better quality solutions (the more parcels per delivery tour, less stops per task, etc.) when the number of total tasks grows. For instance, the average number of tasks per delivery tour increases from 4.60 to 4.85 when the tasks increase from 100 to 200.

Table 6 compares the total distance is made by the crowdsourced delivery system and the traditional delivery system. The percentages in the table demonstrate the fraction of the total distance obtained by the crowdsourced delivery over the total distance obtained by the traditional system. For this benchmark, the base case parameters (see Section 7 and table 1) is used. To further extend the comparison, we also add different task-driver combinations whose ratios are different than 1.

The first conclusion from Table 6 is that the total distance travelled is less for the crowdsourced system than for the traditional system. The results present that the crowdsourced system outperforms from 8% to 25% depending on the instance. This crucial conclusion reveals that it is possible to design a crowdsourced systems that is more cost-efficient than the traditional delivery systems, particularly for the fast delivery services. Table 6 also demonstrates that the crowdsourced system gains extra benefit when the drivers outnumbers the

parcels.

We have already shown that the crowdsourced system is affected positively by the economy of scale; i.e., while both number of tasks and drivers increase in a same amount, the crowdsourced system performs better. Surprisingly, we can also see a similar pattern in the benchmark. Although the economy of scale also holds for traditional delivery systems, the crowdsourced delivery system gains slightly more benefit from it. As a result, with the increasing number of participation, the relative total cost decreases in favour of the crowdsourced delivery systems.

# 8 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 flows in urban areas. In order to represent participants, we present the ad-hoc drivers who are willing to make a small detour in exchange for a small compensation. The problem is formulated 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. The re-usage of the jobs throughout the rolling horizon framework enables us to solve large instances in a dynamic setting.

We also investigate the viability of a P2P crowdsourced delivery platform. We test the performance of the platform with a simulation study based on the city of Rotterdam, the Netherlands. 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 traditional system where all deliveries are made by dedicated drivers. The results reveal that the performance of the crowdsourced delivery system over the traditional system is encouraging. With considerable amount of participation, the crowdsourced delivery system generates better solution in terms of the total distance has to be made to cover all tasks.

In this problem setting, we consider the back-up vehicle as a final option, thus we do

not allow them to make multiple deliveries at the same time. This setting is economically logical when the system has an enough number of ad-hoc drivers. However, particularly in the start-up phase of these type platforms, the usage of dedicated vehicles, which are able to make multiple deliveries, can be not only cost efficient but also a way to keep the service on a certain level. Therefore a problem in which both ad-hoc drivers and dedicated drivers can make multiple deliveries and their synchronisations is an interesting extension of the current study. Further, a future work might consider the information about the uncertainty of the arrival of delivery tasks and drivers. When the arrival pattern of the announcements is taken into a consideration, anticipation methods might be beneficial.

# References


Agatz, N., A. Erera, M. Savelsbergh, and X. Wang (2012). Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research 223*(2), 295–303.

Agatz, N. A., A. L. Erera, M. W. Savelsbergh, and X. Wang (2011). Dynamic ride-sharing: A simulation study in metro Atlanta. *Transportation Research Part B: Methodological 45*(9), 1450–1464.

Archetti, C., M. Savelsbergh, and G. Speranza (2015). The vehicle routing problem with occasional drivers. *Working paper, University of Brescia*.

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

Berbeglia, G., J.-F. Cordeau, I. Gribkovskaia, and G. Laporte (2007). Static pickup and delivery problems: a classification scheme and survey. *Top 15*(1), 1–31.

Berbeglia, G., J.-F. Cordeau, and G. Laporte (2010). Dynamic pickup and delivery problems. *European journal of operational research 202*(1), 8–15.

Einav, L., C. Farronato, and J. Levin (2015). Peer-to-peer markets. Technical report, National Bureau of Economic Research.

Fatnassi, E., J. Chaouachi, and W. 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.

Fung, Brian (2015). Why amazon drone delivery could wind up being the anti-costco.

Furuhata, M., M. Dessouky, F. Ordóñez, M.-E. Brunet, X. Wang, and S. Koenig (2013). Ridesharing: The state-of-the-art and future directions. *Transportation Research Part B: Methodological 57*, 28–46.

Ghilas, V., E. Demir, and T. Van Woensel (2013). Integrating passenger and freight transportation: Model formulation and insights. *Working paper, Technical University of Eindhoven*.

Klapp, M., A. Erera, and A. Toriello (2015). The one-dimensional dynamic dispatch waves problem. *http://www.optimization-online.org*.

Lee, A. and M. 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, B., D. Krushinsky, H. A. Reijers, and T. Van Woensel (2014). The share-a-ride problem: People and parcels sharing taxis. *European Journal of Operational Research 238*(1), 31–40.

Li, B., D. Krushinsky, T. Van Woensel, and H. A. Reijers (2016). An adaptive large neighborhood search heuristic for the share-a-ride problem. *Computers & Operations Research 66*, 170–180.

Masson, R., A. Trentini, F. Lehuédé, N. Malhéné, O. Péton, and H. Tlahig (2014). Optimization of a city logistics transportation system with mixed passengers and goods. *EURO Journal on Transportation and Logistics*, 1–29.

Morphy, E. (2014). About Walmart's idea to crowdsource its same-day delivery service. *Forbes*.

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

Stiglic, M., N. Agatz, M. Savelsbergh, and M. Gradisar (2015). The benefits of meeting points in ride-sharing systems. *Transportation Research Part B: Methodological 82*, 36 – 53.

Voccia, S. A., A. M. Campbell, and B. W. Thomas (2015). The same-day delivery problem for online purchaces. *Working paper, University of Iowa*.

# 9 Appendix A

We can formulate the TSP-TWPC 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 $k$. The route always starts from the origin of the driver and ends at his destination. Let $N^P$ be a set of nodes that corresponds to the origins and destinations of the all tasks $P$ and let $N$ be set of all 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 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, mixed integer problem can be formulated as follows:

$$\min \sum_{i,j \in N} c_{ij} x_{ij} - c_{o_k,d_k} \tag{4}$$

subject to

$$\sum_{j \in N} x_{i,j} = 1, \qquad \forall i \in N^P, \tag{5}$$

$$\sum_{j \in N^p} x_{o_k,j} = 1, \tag{6}$$

$$\sum_{i \in N^p} x_{i,d_k} = 1, \tag{7}$$

$$\sum_{j \in N} x_{ij} - \sum_{j \in N} x_{ji} = 0, \qquad \forall i \in N^p, \tag{8}$$

$$\sum_{i,j \in N} I_{pl(i) \neq pl(j)} x_{ij} \leq Q_k, \tag{9}$$

$$B[i+n] \geq t_{i,i+n} + B[i], \qquad \forall i \in N^{P^+}, \tag{10}$$

$$B_j \geq B_i + t_{ij} - M(1 - x_{ij}) \qquad \forall i,j \in N, \tag{11}$$

$$e_i \leq B_i \leq l_i, \qquad \forall i \in N, \tag{12}$$

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

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

The objective 4 is to minimize total travel cost to serve all delivery task by the driver. Constraint 5 ensures that each task is served exactly once. Constraints 6 and 7 make sure that the driver starts at his origin and ends at his destination. Equations 8 represent the flow conservation constraints. Constraint 9 ensures the maximum stop per driver. In constraint 9, the indicator function $I : L^2 \mapsto 0, 1$ controls the number of different physical locations that the driver visits in his tour. Constraints 10 ensure the precedence relations between pickup and delivery points. Constraints 11 and 12 represent the time window constraints.