A decomposition Algorithm to solve the multi-hop peer-to-peer ride-matching problem

Neda Masoud
Ph.D. Candidate
Department of Civil and Environmental Engineering and
Institute of Transportation Studies
University of California, Irvine
Irvine, CA, USA, 92697
nmasoud@uci.edu

R. Jayakrishnan
Professor
Department of Civil and Environmental Engineering and
Institute of Transportation Studies
University of California, Irvine
Irvine, CA, USA, 92697
rjayakri@uci.edu

Revision Submitted: July 31, 2014
Word Count: 5550
Tables and Figures: 3 Table + 2 Figures = 1750 words
Total Word Count: 7,300

To be considered for Presentation in the 94th TRB Annual Meeting only

ABSTRACT

In this paper, we present a formulation of the multi-hop many-to-many Peer-to-Peer ride-matching problem, found in shared-ride applications. A many-to-many problem is one in which a rider can travel with multiple drivers, and a driver can carry multiple riders. We propose a pre-processing procedure to reduce the size of the problem. Furthermore, we devise a decomposition algorithm to solve the original ride-matching problem to optimality by means of solving multiple smaller problems. Finally, we demonstrate the computational efficiency of the proposed algorithm by solving randomly generated instances of the problem.

INTRODUCTION AND LITERATURE REVIEW

Recent advances in communications technology coupled with the increasing environmental concerns, road congestion, and the high cost of vehicle ownership has directed more attention to the opportunity cost of empty seats travelling throughout the transportation networks every day. Peer-to-peer (P2P) ridesharing is a good way of using the existing capacity on the roads to address the increasing demand for transportation.

Figure 1 shows the average vehicle occupancy in the US in 2010 by trip purpose. The fact that out of an average of 4 seats available in a vehicle only 1.7 is being actually used, suggests a great potential for ridesharing. This potential was recognized by the US congress in June 2012. The section 1501 of the Moving Ahead for Progress in the 21st Century (MAP-21) transportation act expanded the definition of "carpooling" to include "real-time ridesharing" as well, making ridesharing eligible for all the federal funding that was previously available only for carpooling projects.

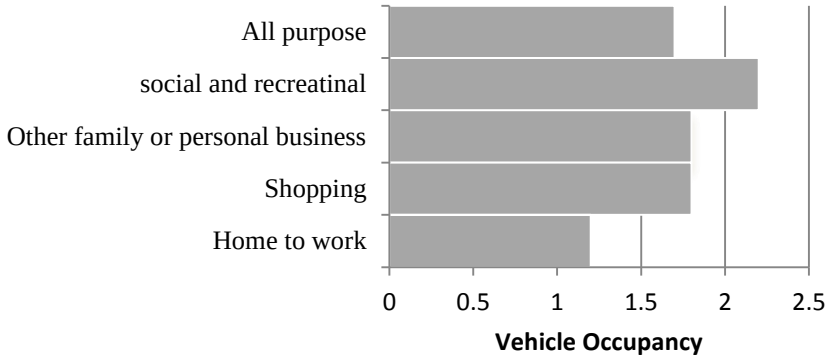


FIGURE 1 Average vehicle occupancy[1] in the US in 2009 for different trip purposes
Source of data: 2009 National Household Travel Survey (NHTS)

In this paper, we define P2P real-time ridesharing to include all one-time rideshares with any type of arrangement, on the fly or pre-arranged, between peer drivers and riders. The term "real-time" emphasizes the capability of the system to make ride-matches in real time. The system finds matches for riders by optimally routing drivers in the network. We define a set of stations in the network where riders can start and end their trips, or switch between vehicles.

A P2P ride-matching algorithm is central to a ridesharing system. A ride-matching problem is a problem of matching riders and drivers in a ridesharing system. If a rider is matched in the system, he/she receives an itinerary of his/her trip including the information on the scheduled route, and drivers to ride with. Drivers receive itineraries that include the schedules to pick up and drop off riders. Following the terminology from *(1)*, we call these itineraries route plans[2].

The P2P ride-matching problem has attracted attention in academia only in the very recent years. Table 1 presents some of the attempts made to formulate the P2P ride-matching problem and/or propose algorithms to solve it. In this table, we have listed some of the characteristics of a flexible ridesharing system, and reviewed and assessed the literature based on these criteria. This summary suggests that the previous attempts each lack at least one key component that could potentially increase the number of successful matches made in the system.

[1] NHTS defines vehicle occupancy as person miles of travel per vehicle mile
[2] *(1)* uses route plan only to refer to the riders' itinerary

Some formulations and algorithms that assume a fixed route for each driver miss the potential matches that could be made by having the system assign routes to drivers. Of course, considering fixed routes for drivers can substantially decrease the computational complexity of the problem, but such savings come at the cost of losing potential matches and thus better system efficiency.

Most of the current work in the literature does not consider multi-hop routes for riders. Multi-hop routes are routes in which riders can switch between drivers. A multi-hop P2P ridesharing system comes in handy especially when P2P ridesharing systems are integrated with transit. A multi-hop system can improve the number of riders' requests that can be satisfied, but increases the complexity of the problem.

There are formulations that solve the ride-matching problem for one rider at a time. These formulations are inadequate, since once a ride-matching problem is solved for a rider, the routes of drivers that construct the rider's route plan are fixed, and this translates into the opportunity cost for subsequent riders.

Finally, none of the proposed algorithms finds an optimal solution to the ride-matching problem, and the proposed mixed integer programming problems (MIP), which have the potential to reach optimal solutions, typically do not consider a time limit for reaching the solution, thus making the formulations to not necessarily be compatible with the requirements of real-time systems.

In this paper, we propose an MIP formulation of the P2P ride-matching problem that contains all the above components. In addition, to ensure that rides can be successfully accomplished, time-dependent travel time matrices are used in this study. Furthermore, the proposed formulation gives drivers the possibility of visiting each station multiple times, if necessary.

In the rest of the paper, we first introduce the ridesharing system to provide the context in which we need to solve the ride-matching problem. The rest of the paper focuses on formulating and solving the ride-matching problem. We start by formulating the P2P ride-matching problem. Since solving this problem directly is computationally expensive, we introduce a pre-processing procedure to limit the size of the problem, and a decomposition algorithm to help solve the problem more efficiently. Finally we talk about the scalability of the problem and its application in real-time.

TABLE 1 Summary of the Previous Work on the Ride-Matching Problem

Author	Year	Fixed Paths	Multi-hop	Multiple Riders	Formulation	Optimal Solution
Ghoseiri *(2)*	2012	Y[3]	Y	Y	Optimization with heuristic solution	N
Herbawi and Weber *(1)*	2011	Y	Y	N	genetic and evolutionary algorithms	N
Febbraro et al. *(3)*	2013	N	N	Y	Optimization	Y
Agatz et al. *(4)*	2009	N	Y	N	Optimization	Y
Herbawi and Weber *(5)*	2012	N	N	Y	Optimization with heuristic solution	N

[3] Allows for short detours, but drivers have to follow the pre-determined set of nodes.

RIDESHARING SYSTEM

The ridesharing system defined in this paper contains a set of participants P who are willing to provide or receive rides in the (near or distant) future. These participants are divided into a set of riders, R, who are looking for a ride, and a set of drivers, D, who are willing to provide rides in return for monetary compensation, or for any other ridesharing incentives such as using HOV lanes, etc.

To facilitate pick-ups and drop-offs, a set of stations,S, are identified in the network. Stations in the ridesharing system described in this paper are pre-specified locations where riders can start their trip, end it, or switch between drivers and/or to and from the transit system. Strategic identification of stations is central to the performance of the system. Lessons learnt from the previous P2P ridesharing systems suggest that it is better for riders to be picked up/dropped off at pre-specified stations, than their homes (or the exact location where their trips start/end) for two reasons *(6)*. First, these locations could be hard to find for drivers, and therefore people might miss their rides. In addition, drivers could have a hard time finding an appropriate location to park their vehicles. Second, some drivers and riders would understandably be reluctant to reveal their home address to others.

All participants when having a ride-share request input into the system their origin and destination stations (OS_p and DS_p respectively), the earliest time they are willing to depart from their origin station, T_p^{ED}, the latest time they have to arrive at their destination T_p^{LA}, and the maximum trip length they can afford T_p^{TL}. The travel time window for each participant is defined as $TW_p = \left[T_p^{ED} \;\; T_p^{LA}\right]$.

Each driver should also determine the capacity of his/her vehicle, C_d. This could simply be the physical capacity of the vehicle, or the maximum number of riders the driver is willing to carry in his/her vehicle at each instant in time. Each rider can specify the maximum number of connections (change of vehicles), V_r, he/she is willing to take to get to his/her destination (Note that the term "connection" used here is essentially identical in nature to the term "transfer" used in the public transit context. We use these terms interchangeably).

We discretize the study time horizon into short time periods, Δt, to allow for using time-dependent travel-time matrices in the system. We keep the set of time periods for each participant p in set T_p. T_p contains all the time periods within the range $TW_p = \left[T_p^{ED} \;\; T_p^{LA}\right]$.

In a system discretized in both time and space, we define a node i, n_i, as a tuple (t_i, s_i), where t_i is the time period one can arrive at/leave s_i. Subsequently, a link, l, is defined as $\left(n_i, n_j\right) = (t_i, s_i, t_j, s_j)$, where t_i is the time period one has to leave s_i, in order to arrive at s_j at time period t_j.

The goal of the ridesharing system is to match riders with drivers. For now, let's assume that drivers leave the route choice to the system. This does not preclude the case of drivers who want to follow their own fixed routes, as those routes can be specified as successive nodes and entered into the formulation as fixed parameters.

Each rider has to provide a deadline by which he/she needs to be informed of any matches made for him/her in the system. A few minutes before this deadline, a ride-matching problem is solved for the rider. This problem includes a closed system of participants, such that the travel time window of no participants outside of the system intersects with the travel time window of any participant inside the system, and the travel time window of all participants inside the system intersects with the travel time window of at least one other participant inside the system. Riders and drivers inside this system are kept in sets R_r and D_r respectively. If there are

drivers in set D_r who are previously matched in the system, but still have empty seats in their vehicles, they are added to the problem with fixed routes. The paths of the drivers who have not yet been matched with any riders will be determined by the ride-matching problem.

We include a dummy driver, d_{dummy}, to the set of drivers, and form the set $D'_r = D_r \cup d_{dummy}$. The dummy driver doesn't have a real origin or destination, nor a travel time window. The motivation behind introducing the dummy driver will be explained in the following section.

PEER-TO-PEER RIDE-MATCHING PROBLEM

The objective of the ride-matching problem is to devise route plans that can take riders to their destinations by optimally routing drivers. Route plans have to comply with their specified maximum number of transfers, and the capacity of drivers' vehicles. The ride-matching problem will devise route plans for the matched riders in the system, and all drivers, matched or not.

The mathematical formulation of the ride-matching problem contains four set of decision variables defined in (1)-(4).

$$X_l^d = \begin{cases} 1 & driver\ d\ travels\ on\ link\ l \\ 0 & otherwise \end{cases} \quad (1)$$

$$X_l^{rd} = \begin{cases} 1 & rider\ r\ travels\ on\ link\ l\ with\ driver\ d \\ 0 & otherwise \end{cases} \quad (2)$$

$$Y_r = \begin{cases} 1 & rider\ r\ is\ matched \\ 0 & othrwise \end{cases} \quad (3)$$

$$U_r^d = \begin{cases} 1 & driver\ d\ constructs\ part\ of\ the\ route\ plan\ for\ rider\ r \\ 0 & otherwise \end{cases} \quad (4)$$

The system of constraints that define the ride-matching problem is presented in (5.2)-(5.18). Constraints (5.2)-(5.4) route drivers in the system. (5.2) direct drivers in the set D_r out of their origin stations, and (5.3) ensure that they end up in their destination stations. (5.4) is the balancing constraint, enforcing that a driver who enters a station in a time period, exits the station in the same time period. Notice that participants might not physically leave a station. Members of set L in the form $(t_i, s, t_i + \Delta t, s)$ represent the situations where a participant is staying at station s for one time period.

Rider r's route plan is determined by the variable X_l^{rd}. A value of 1 for this variable indicates that the rider has travelled on link l in driver d's vehicle. By definition, this variable implies that a rider should always be accompanied by a driver. However, in reality a rider doesn't need to be accompanied when he/she is taking a link in the form $(t_i, s, t_i + \Delta t, s)$, i.e. lingering at a station, either to start a trip, end it, or make a transfer. To incorporate this element into the math problem, the dummy driver has been introduced into the formulation. As mentioned before, the dummy driver does not have a real origin or destination in the network. The set of links for the dummy driver is also different from the set L used by the participants. We define the set of links for the dummy driver as $L_{d_{dummy}} = \{(t_i, s, t_i + \Delta t, s), \forall t_i \in T, \forall s \in S\}$. This set includes all the links that indicate staying at a station for one time period. Constraints (5.5) ensure that $X_l^{d_{dummy}} = 1, \forall l \in L_{dummy}$, so that these links can be used by riders whenever necessary. (5.6) limit the total travel time by each driver.

Constraints (5.7)-(5.9) route riders in the system, and are analogous to (5.2)-(5.4), except for a small variation. While drivers, matched or not, will receive a route plan from the system, this is not the case for the riders. Only riders who are matched in the system will receive route plans. This difference between routing of riders and drivers is reflected in the formulation by

replacing 1 on the right hand side of constraints (5.2)-(5.3) by Y_r in (5.7)-(5.8). (5.10) set a limit on the total travel time by each rider.

Constraints (5.11) ensure that riders are accompanied by drivers throughout their trips. Notice that this constraint set precludes the links in which $s_i = s_j$. This type of links are taken care of in (5.5) by the dummy driver.

(5.12) set limits on the capacity of vehicles. (5.13) register drivers who collectively construct each rider's route plan (refer to proposition 1). (5.14) restrict the number of transfers by each rider (refer to proposition 2).

(5.15)-(5.18) determine the type of the decision variables. While X_l^d and U_d^r are binary variables, the binary condition can be relaxed for X_l^{rd} and Y_r (proposition 3).

(5.1) shows the objective function of the problem. The ride-matching problem can have different objectives, ranging from maximizing profits to minimizing the total miles/hours travelled in the system. This objective can vary depending on the type of the agency (public or private), and the level of acceptance of the system in the community. For a ridesharing system at its infancy, it sounds logical to try to maximize the number of satisfied rides in the system, and at the same time minimize the number of transfers to make the riders as comfortable as possible. We use this objective for the ridesharing system in this paper. The first term in (5.1) minimizes the total number of connections in the system, and the second term maximizes the total number of served riders. The relative importance of each term can be set using weights in the objective function. Here, we set the weight for the number of satisfied riders dependent on the rider IDs. Riders whose departure times are farther away should have lower weights, since they might have other opportunities as drivers keep joining the system.

$$Minimize\ W_T \sum_{\substack{r\in R_r \\ d\in D_r}} U_r^d - W_s^r \sum_{r\in R_r} Y_r \tag{5.1}$$

$$\sum_{\substack{l\in L: \\ s_i=OS_d}} X_l - \sum_{\substack{l\in L: \\ s_j=OS_d}} X_l = 1 \qquad \forall d \in D_r \tag{5.2}$$

$$\sum_{\substack{l\in L \\ s_j=DS_d}} X_l = 1 \qquad \forall d \in D_r \tag{5.3}$$

$$\sum_{\substack{t_i,s_i: \\ l=(t_i,s_i,t,s)\in L}} X_l^d = \sum_{\substack{t_j,s_j: \\ l=(t,s,t_j,s_j)\in L}} X_l^d \qquad \begin{matrix}\forall d \in D_r \\ \forall t \in T_d, \\ \forall s \in S - \{OS_d, DS_d\}\end{matrix} \tag{5.4}$$

$$X_l^{d_{dummy}} = 1 \qquad \forall l \in L_{d_{dummy}} \tag{5.5}$$

$$\sum_{l\in L} (t_j - t_i) X_l^d \le T_p^{TL} \qquad \forall d \in D_r \tag{5.6}$$

$$\sum_{d\in D_r'} \sum_{\substack{l\in L: \\ s_i=OS_r}} X_l^{rd} - \sum_{d\in D_r'} \sum_{\substack{l\in L: \\ s_j=OS_r}} X_l^{rd} = Y_r \qquad \forall r \in R_r \tag{5.7}$$

$$\sum_{d\in D'_r}\sum_{\substack{l\in L:\\ s_j=DS_r}} X_l^{rd} = Y_r \qquad \forall r \in R_r \qquad (5.8)$$

$$\sum_{d\in D'_r}\sum_{\substack{t_i,s_i:\\ l=(t_i,s_i,t,s)\in L}} X_l^{rd} = \sum_{d\in D'_r}\sum_{\substack{t_j,s_j:\\ l=(t,s,t_j,s_j)\in L}} X_l^{rd} \qquad \forall r \in R_r, \forall t \in T_r, \ \forall s \in S - \{OS_r, DS_r\} \qquad (5.9)$$

$$\sum_{d\in D'_r}\sum_{l\in L}(t_j - t_i)X_l^{rd} \le T_p^{TL} \qquad \forall r \in R_r \qquad (5.10)$$

$$X_l^{rd} \le X_l^d \qquad \forall r \in R_r, \forall d \in D_r, \ \forall l \in L: s_i \neq s_j \qquad (5.11)$$

$$\sum_{r\in R_r} X_l^{rd} \le C_d\, X_l^d \qquad \forall d \in D_d, \ \forall l \in L \qquad (5.12)$$

$$U_r^d \ge X_l^{rd} \qquad \forall r \in R_r, \forall d \in D_r, \ \forall l \in L \qquad (5.13)$$

$$\sum_{d\in D_r} U_r^d - 1 \le V_r \qquad \forall r \in R_r \qquad (5.14)$$

$$X_l^d \in \{0,1\} \qquad \forall d \in D_d, \ \forall l \in L_d \qquad (5.15)$$

$$0 \le X_l^{rd} \le 1 \qquad \forall r \in R_r, \forall d \in D_r, \ \forall l \in L_{rd} \qquad (5.16)$$

$$U_r^d \in \{0,1\} \qquad \forall r \in R_r, \forall d \in D_r \qquad (5.17)$$

$$0 \le Y_r \le 1 \qquad \forall r \in R_r \qquad (5.18)$$

Solving the optimization problem in (5) is computationally prohibitive even for small instances of the problem. Therefore, in its original form, the ride-matching problem in (5) is not appropriate for real-time applications. In the next section, we introduce a pre-processing procedure that limits the input to the optimization problem. Next, we propose a disaggregation algorithm that attempts to solve the original ride-matching problem by means of solving multiple smaller problems.

PRE-PROCESSING PROCEDURE

The goal of the pre-processing procedure is to limit the number of links accessible by each participant, and hence the input to the mixed integer ride-matching problem. As a reminder, we present a link, l, as a 4-tuple $(t_{\mathrm{i}}, s_{\mathrm{i}}, t_{\mathrm{j}}, s_{\mathrm{j}})$. Each participant can potentially reach any station in the network at any time period, making the size of the set of links, L, as large as $O(|T|^2|S|^2)$, where $|T|$ is the number of time periods in the study time horizon, and $|S|$ is the number of stations in the network.

It is evident that in most cases participants won't have access to all members of set L, due to the spatiotemporal constraints enforced by their origin and destination stations, maximum acceptable travel times, and travel time windows. We use this information to construct the set of links accessible to riders and drivers, designated by L_r and L_d respectively.

The origin and destination stations, maximum acceptable travel times, and travel time windows of participants can be used to define a region in the network in the form of an ellipse, inside which participants have a higher degree of space proximity, i.e. the percentage of accessible stations within this region is at least as high as the same percentage within the entire network. We call the region inside and on the circumference of the ellipse associated with participant p the reduced graph of the participant, denoted by G_p.The focal points of the ellipse are the participant's origin and destination stations. The length of the major axes between the focal points is the straight distance between the origin and destination stations, and the transverse diameter of the ellipse is an upper-bound on the distance that can be travelled by the participant in T_p^{TL} time units, during the participant's time window.

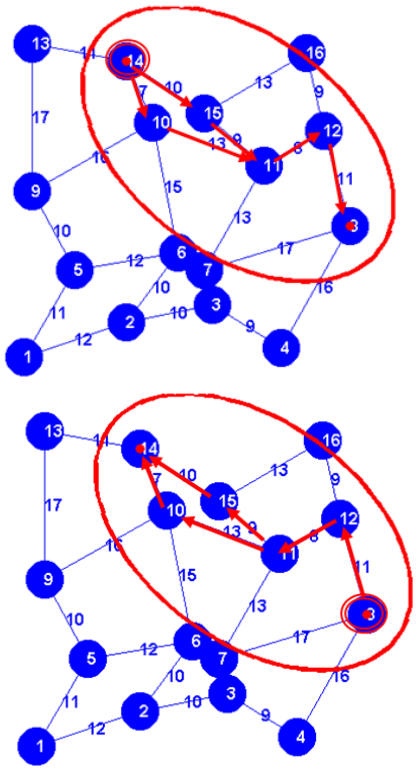
After the reduced graphs are identified, we use the following algorithm to construct sets L_r and L_d. The algorithm finds the set of links in two steps: a forward movement followed by a backward movement. In the forward movement, the algorithm starts with the origin station, assuming that the participant can leave this station in time intervals within the window $[T_p^{ED} \quad T_p^{LA} - tt_{OS_p,DS_p}]$, where tt_{OS_p,DS_p} is the shortest path travel time between the origin and destination stations of participant p. tt_{OS_p,DS_p} is calculated based on the static travel time matrix T_{static}. To ensure no feasible links are cut off, T_{static} should contain an underestimation of the link travel times, say for example the travel times during non-peak hours.

After identifying the set of feasible time periods for the origin station, outgoing links from the origin station whose end nodes are inside the reduced graph are identified. The set of time periods for these stations is determined based on the set of time periods for the origin station and the time-dependent travel times on the links. The same procedure repeats until the destination station is reached. Figure 2 demonstrates an example of the forward movement for a participant who is travelling from station 14 to station 8, with $TW_p = [1\ 40]$ (in minutes), travel time budget of 40 minutes, and shortest path travel time of 37 minutes. The time periods are assumed to be 1 minute each. Links travel times (in minutes) are shown on the links of the graph. It is assumed that the travel time remains constant on each link. Note that this assumption is made only for simplicity, and using time-dependent travel times would be straightforward, as the formulation already has a multiple period structure. The set of time periods within which each station is accessible is computed during the forward movement, and presented in the figure. These time periods are not final, however, and have to be refined using the backward movement.

The backward movement simply scans through the table created by the forward movement and refines it by removing some of the time intervals. In the example shown in Figure 2, since the latest arrival time is at $\Delta t = 40$, time periods 40 and 41 should be removed from the set of time periods for the destination station. Tracking back the stations from the destination to the origin, time periods for the other stations that has led to the excessive time periods at the destination station are removed. In the example in Figure 2, after completing the backward movement, the set of links, L_p, are derived and listed in the figure.

The last step of the pre-processing procedure is to find the drivers who have spatiotemporal proximity to each rider. This can be accomplished for each rider r by comparing members of the set L_r with members of the sets $L_d, \forall d \in D_r$: $(TW_r \cap TW_d) = \emptyset$. For each driver

d, if $L_r \cap L_d \neq \emptyset$, tuple (r, d) will be added to the set M. In addition, a set L_{rd} will be constructed containing all the links in $L_r \cap L_d$. Furthermore, we add tuples $(r, d_{dummy}), \forall r \in R_r$ to set M, and set $L_{rd_{dummy}} = L_{d_{dummy}}, \forall r \in R_r$.



Forward Movement

Station	Pred.	Time Intervals
14	-	1,2,3
15	14	11,12,13
10	14	8,9,10
11	15	20,21,22
11	10	21,22,23
12	11	28,29,30,31
8	12	30,40,41,42

Backward Movement

Station	Pred.	Time Intervals
8	12	39,40
12	11	28,29
11	10	21
11	15	20
10	14	8
15	14	11
14	-	1

$$L_p = \{(1,14,11,15), (1,14,8,10), (11,15,20,11), (8,10,21,11), (20,11,28,12), (20,11,,21,11), (21,11,29,12), (28,12,39,8), (39,8,40,8), (29,12,40,8)\}$$

FIGURE 2 An example of a forward and backward movement for a participant with $OS_p = 14, DS_p = 3,\ \Delta t = 1\ min, TW_p = [1\ 40]\ min$, and $T_p^{TL} = 40\ min$

DECOMPOSITION ALGORITHM

The decomposition algorithm described in this section attempts to solve the original problem by solving a number of sub-problems that are easier to solve. The flowchart of this algorithm is displayed in Figure 3. The basic idea is that in each iteration the algorithm solves a number of sub-problems that can represent the entire system. If the solutions to the sub-problems do not have any conflicts, the algorithm is terminated and the combination of solutions to the sub-problems can give the global optimal for the original problem. Each sub-problem includes a number of riders kept in the set R_{rs}, and a number of drivers kept in the set $D_{rs} = \{\forall d | r \in R_{rs} \cap (r, d) \in M\}$. We identify sub-problems by the set of riders in the sub-problem.

The algorithm starts by solving R_r sub-problems, each including one of the riders in the set R_r, and all the drivers d such that $(r, d) \in M$. In the case of there being no conflicts between the solutions, the solution to the original problem is readily available. This happens if each rider

is matched with a different driver, or if multiple riders are matched with the same driver and the driver is capable of performing all pick-up and drop-off assignments for the assigned riders.

In each iteration, in the case of there being conflicts between solutions of the sub-problems in the previous iteration, we form the "applicable" sub-problems. An "applicable" sub-problem includes: (i) a group of riders from the last iteration's sub-problems with identical driver assignment (if assignments conflict in time or space), and (ii) sub-problems in the previous iterations from which riders are detached (with the remaining set of riders). For example, assume that at some iteration we have two sub-problems $\{r_1, r_2\}$ and $\{r_3, r_4\}$. The solutions to these sub-problems are $\{r_1: d_1, r_2: d_2\}$ and $\{r_3: d_1, r_4: d_3\}$. Here d_1 is assigned to both r_1 and r_3 but through different routes. Therefore, these two riders form a new sub-problem $\{r_1, r_3\}$. Since the two previous sub-problems have each lost a rider, they have to form new sub-problems as well, because the solution to them might not be optimal anymore. Hence in the new iteration we have 3 sub-problems: $\{r_1, r_3\}$, $\{r_2\}$, and $\{r_4\}$. However, not all these sub-problems have to be actually solved, as we will see later.

After the new applicable sub-problems are formed, first we have to check for the loops between iterations. For example, consider the case where the first iteration includes the $\{r_1, r_2\}$ sub-problem, the second iteration includes the $\{r_2, r_3\}$ sub-problem, and the following iteration includes the $\{r_1, r_2\}$ sub-problem again. If we keep proceeding, we will be caught in a loop. To come out of this loop, we form an "intermediate" sub-problem. The "intermediate" sub-problem combines the sub-problems involved in the loop. In the example above the intermediate sub-problem is $\{r_1, r_2, r_3\}$.

After all the new sub-problems are determined, a decision has to be made on whether the sub-problems need to be solved or not. Sub-problems that need to be solved are called "active" sub-problems. These sub-problems are the ones whose optimal solutions cannot be readily obtained from the previous solutions. For example, assume that we have a sub-problem $\{r_1, r_2\}$. From the initialization of the algorithm, we know that matches $\{r_1: d_1\}$ and $\{r_2: d_2\}$ are optimal. Since the separate solutions for these riders don't have any conflicts, we can readily conclude that the solution to the sub-problem is $\{r_1: d_1, r_2: d_2\}$. So $\{r_1, r_2\}$ is not an active sub-problem.

Note that while using this disaggregation algorithm, a large problem could be solved in the first iteration, or we might end up solving multiple problems before having to solve the original problem in the last iteration. The main merit of this algorithm is that sub-problems during each iteration can be solved in parallel.

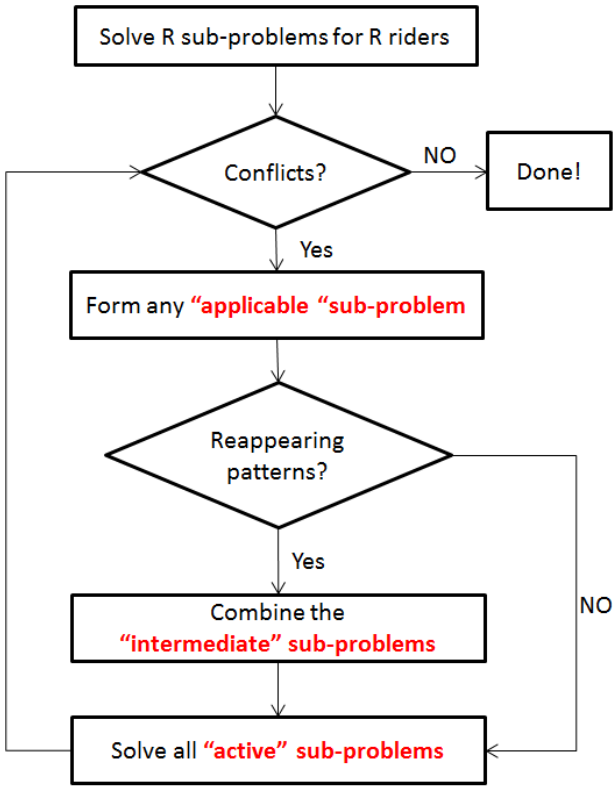


FIGURE 3 The disaggregated problems algorithm

ILLUSTRATIVE EXAMPLE

Assume that a ridesharing system has 6 riders and 4 drivers, and all drivers have spatiotemporal proximity with all riders. The iterations of the disaggregated algorithm used to match the participants are displayed in Table 2. The active sub-problems during each iteration are displayed in blue. The problems whose solutions won't change throughout the entire decomposition algorithm are displayed in green.

In iteration 1, each rider constitutes an active sub-problem. The solutions show that riders 1, 3 and 6 all have driver 1 in their solution, but in conflicting paths. Therefore, an active sub-problem of $\{r_1, r_3, r_6\}$ is formed and solved in the second iteration. Also, since rider 2 was not able to find any matches even without the competition from other riders, he will not be able to find a match in the current configuration of the system. Sub-problems r_2 and r_5 are not active sup-problems and their solutions are readily available.

The solution to the active sub-problem $\{r_1, r_3, r_6\}$ in iteration 2 shows that the optimal matches for riders 5 and 6 are in conflict (they are both matched with driver 2, but through different paths). So they form the sub-problem $\{r_5, r_6\}$. Also, since rider 6 was removed from the sub-problem $\{r_1, r_3, r_6\}$, the solution obtained for this sub-problem for riders 1 and 3 might not be optimal anymore. Therefore, a new sub-problem $\{r_1, r_3\}$ is formed. However, not both these sub-problems are active. The optimal match for rider 5 is driver 2, and the optimal match for rider 6 is driver 1. Since these two don't have any conflicts, the solution to the $\{r_5, r_6\}$ sub-problem is readily available.

The only active sub-problem in iteration 3 is $\{r_1, r_3\}$. The solution to this sub-problem suggests that two new sub-problems $\{r_1, r_3, r_6\}$ and $\{r_5\}$ need to be formed, which along with two sub-problems $\{r_4\}$ and $\{r_2\}$ should constitute the set of sub-problems for iteration 4. However, we had the exact same set of sub-problems iteration 2. Therefore, in order to avoid getting caught in a loop, a new (intermediate) sub-problem $\{r_1, r_3, r_6, r_5\}$ is formed in iteration 4. After solving this sub-problem, there are no more conflicts. So the global solution to the original problem is obtained in iteration 4. A total of 9 sub-problems had to be solved for this solution to be obtained. However, in iteration 1, all the 6 active sub-problems could be solved simultaneously.

TABLE 2 Iterations of the Disaggregation Algorithm

Iteration 1	r_1 d_1	r_2 -	r_3 d_1, d_2	r_4 d_3	r_5 d_2	r_6 d_1
Iteration 2	r_1 d_1	r_3 d_1, d_4	r_6 d_2	r_2 -	r_4 d_3	r_5 d_2
Iteration 3	r_1 d_1	r_3 d_1, d_4	r_6 d_1	r_5 d_2	r_2 -	r_4 d_3
Iteration 4	r_1 d_1	r_3 d_1, d_4	r_6 d_1	r_5 d_2	r_2 -	r_4 d_3

It is possible to solve another version of the algorithm which is easier to implement, but may take longer to solve. In this simplified version, if any two riders in two sub-problems have conflicts, the entire sub-problems are combined in the following iteration. This will lead to potentially fewer number of iterations, but larger sub-problems to be solved in each iteration. This algorithm is applied to the example above. Here, after solving the active sub-problem $\{r_1, r_3, r_6\}$ in iteration 2, and studying the solutions to all sub-problems, it turns out that rider 6 and rider 5 are both matched with driver 2, but through conflicting paths. Therefore, in the next iteration the two sub-problems $\{r_1, r_3, r_6\}$ and $\{r_5\}$ are combined. In this particular example using the simplified version of the algorithm leads to reaching the optimal solution in fewer number of iterations, and less amount of time, since we are skipping iteration 3 in table 4. However, this is not a typical behavior of this algorithm.

TABLE 3 Iterations of the Simplified Disaggregation Algorithm

Iteration 1	r_1 d_1	r_2 -	r_3 d_1, d_2	r_4 d_3	r_5 d_2	r_6 d_1
Iteration 2	r_1 d_1	r_3 d_1, d_4	r_6 d_2	r_2 -	r_4 d_3	r_5 d_2
Iteration 3	r_1 d_1	r_3 d_1, d_4	r_6 d_1	r_5 d_2	r_2 -	r_4 d_3

REVISED VERSION of THE P2P RIDE-MATCHING PROBLEM

After performing the pre-processing procedure, each sub-problem in the decomposition algorithm can be solved using the system of equations (6). The problem is very similar to the problem in (5), with two main differences: (i) sets in (6) are more refined owing to both the pre-

processing procedure and the decomposition algorithm. (ii) Constraints (5.6) and (5.10) are now redundant, since the requirement to not exceed the maximum trip length is met while forming the reduced graphs and performing the forward and backward movements in the pre-processing procedure.

$$Minimize\ W_T \sum_{\substack{r\in R_{rs}\\ d:(r,d)\in M}} U_r^d - W_s^r \sum_{r\in R_{rs}} Y_r \tag{6.1}$$

$$\sum_{\substack{l\in L_d:\\ s_i=OS_d}} X_l - \sum_{\substack{l\in L_d:\\ s_j=OS_d}} X_l = 1 \qquad \forall d \in D_{rs} \tag{6.2}$$

$$\sum_{\substack{l\in L_d\\ s_j=DS_d}} X_l = 1 \qquad \forall d \in D_{rs} \tag{6.3}$$

$$\sum_{\substack{t_i,s_i:\\ l=(t_i,s_i,t,s)\in L_d}} X_l^d = \sum_{\substack{t_j,s_j:\\ l=(t,s,t_j,s_j)\in L_d}} X_l^d \qquad \begin{matrix}\forall d \in D_{rs}\\ \forall t \in T_d,\\ \forall s \in G_d - \{OS_d, DS_d\}\end{matrix} \tag{6.4}$$

$$X_l^{d_{dummy}} = 1 \qquad \forall l \in L_{d_{dummy}} \tag{6.5}$$

$$\sum_{\substack{d\in D'_{rs}:\\ (r,d)\in M}} \sum_{\substack{l\in L_{rd}:\\ s_i=OS_r}} X_l^{rd} - \sum_{\substack{d\in D'_{rs}:\\ (r,d)\in M}} \sum_{\substack{l\in L_{rd}:\\ s_j=OS_r}} X_l^{rd} = Y_r \qquad \forall r \in R_{rs} \tag{6.6}$$

$$\sum_{\substack{d\in D'_{rs}:\\ (r,d)\in M}} \sum_{\substack{l\in L_{rd}:\\ s_j=DS_r}} X_l^{rd} = Y_r \qquad \forall r \in R_{rs} \tag{6.7}$$

$$\sum_{\substack{d\in D'_{rs}:\\ (r,d)\in M}} \sum_{\substack{t_i,s_i:\\ l=(t_i,s_i,t,s)\in L_{rd}}} X_l^{rd} = \sum_{d\in D'_{rs}} \sum_{\substack{t_j,s_j:\\ l=(t,s,t_j,s_j)\in L_{rd}}} X_l^{rd} \qquad \begin{matrix}\forall r \in R_r, \forall t \in T_r,\\ \forall s \in G_r - \{OS_r, DS_r\}\end{matrix} \tag{6.8}$$

$$X_l^{rd} \le X_l^d \qquad \begin{matrix}\forall (r,d) \in M\\ \forall l \in L_{rd}: s_i \ne s_j\end{matrix} \tag{6.9}$$

$$\sum_{\substack{r\in R_{rs}:\\ (r,d)\in M, l\in L_{rd}}} X_l^{rd} \le C_d\, X_l^d \qquad \begin{matrix}\forall d \in D_{ds},\\ \forall l \in L_d\end{matrix} \tag{6.10}$$

$$U_r^d \ge X_l^{rd} \qquad \begin{matrix}\forall (r,d) \in M\\ \forall l \in L_{rd}\end{matrix} \tag{6.11}$$

$$\sum_{\substack{d\in D_{rs}:\\ (r,d)\in M}} U_r^d - 1 \le V_r \qquad \forall r \in R_{rs} \tag{6.12}$$

$$X_l^d \in \{0,1\} \qquad \begin{matrix}\forall d \in D_{ds},\\ \forall l \in L_d\end{matrix} \tag{6.13}$$

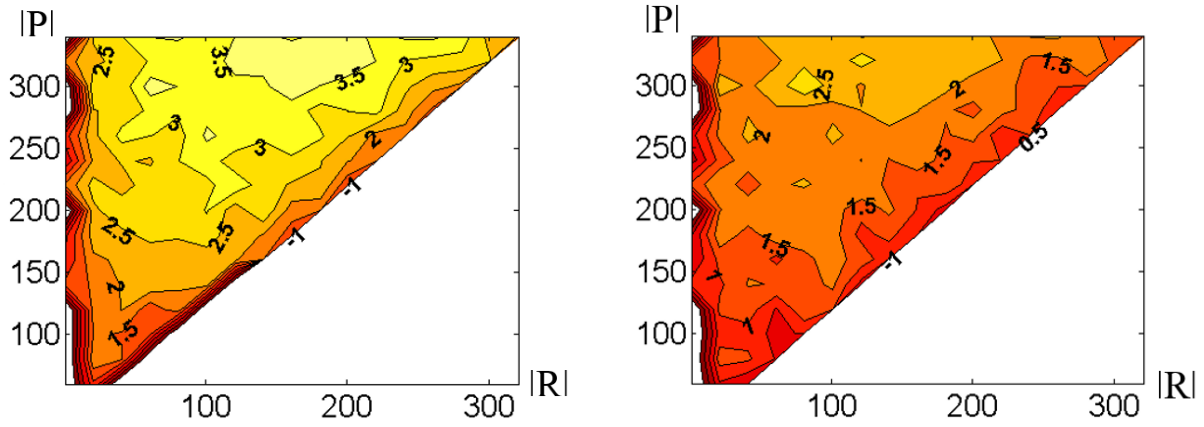
$$0 \le X_l^{rd} \le 1 \qquad \forall (r,d) \in M,\ \forall l \in L_{rd} \qquad (6.14)$$

$$U_r^d \in \{0,1\} \qquad \forall (r,d) \in M \qquad (6.15)$$

$$0 \le Y_r \le 1 \qquad \forall r \in R_{rs}$$

To evaluate the performance of the proposed decomposition algorithm, we solved multiple random instances of the ride-matching problem in a randomly-generated grid network with 81 stations, with and without the decomposition algorithm. The number of participants $|P|$ in the generated problems varied between 60 and 300, and the number of riders $|R|$ between 1 and $|P| - 10$. The numerical tests were performed on a PC with Core i7 3 GHz and 8GB of RAM.

We took the logarithm of the solution times in base 10, and plotted the solution time contour lines in Figure 4. The logarithms are shown simply to display numbers that are easier to see in the contour plots. As an example, for a case of about 175 participants, with about 50 riders, the left-side plot indicates a 2,5 which is $10^{2.5}$ = 316 seconds of computation time. The same case shows 1.5 on the right-side plot ($10^{1.5}$= 31.6) indicating a 10-fold benefit in computational time. The two plots quite clearly and succinctly show that solution time savings in obtained by the decomposition algorithm are in order of 10. In addition, no branch and bounds were required to solve any of the problem instances.



4(A) Log In Base 10 of Solution Times Using the MILP Formulation

4(b) Log In Base 10 of Solution Times Using the Decomposition Algorithm

FIGURE 4 Comparing Solution Times of Problem Instances With and Without the Decomposition Algorithm

In addition to the relative savings in computational times, it is also important to look at the actual computation times. As described above, the solution time under the decomposition scheme was a mere 30 seconds to solve a fairly decent sized and realistic ride-matching problem of nearly 175 drivers and 50 riders in an 81 node network.

The problem can be further decomposed in rolling-horizon solution schemes in real time as well, when for instance 100 or 200 participants are added to an existing system every few minutes and a solution is found within a time period of the order of 10s of seconds with current computational capabilities, which makes the algorithm useful, with appropriate modifications even for systems of 1000s for vehicles. Thus, subject to further studies on implementation aspects which are beyond the scope of this paper, the algorithm shows excellent promise for the ride-matching problems of several different types of real-time ride share systems that are currently emerging.

One important point to keep in mind is that the solutions rendered by the decomposition algorithm are optimal, i.e. the same solutions obtained from the MILP formulation of the problem.

Conclusion

In this paper, we proposed a mixed integer linear optimization problem for the P2P ride-matching problem. Despite the fact that the formulation is a mixed integer problem, the structure of the formulation limits the necessity for branching only to very specific cases. In addition, although in the formulation presented in this paper we used a very specific objective function, the decision variables used in the formulation allow for implementing a wide variety of objective functions.

We devised algorithms to improve the solution time of the problem, maintaining the optimality of the solutions. The computational savings due to these algorithms are considerable, and make the formulation more suitable for real-time applications.

REFERENCES

(1) Herbawi, W., and M. Weber. Comparison of Multiobjective Evolutionary Algorithm for Solving The Multiobjective Route Planning in Dynamic Multi-hop Ridesharing. Proceedings of the 11th European conference on Evolutionary computation in combinatorial optimization, Torino, Italy, 2011, pp. 84-95.

(2) Ghoseiri, K. DYNAMIC RIDESHARE OPTIMIZED MATCHING PROBLEM, Dissertation at University of Maryland, 2013.

(3) Di Febbraro, A., E. Gattorna, and N. Sacco. Optimizing Dynamic Ride-Sharing Systems. Presented in the TRB 2013 annual meeting. 2013.

(4) Agatz, N., A. Erera, M. Savelsbergh, and X. Wang. Sustainable Passenger Transportation: Dynamic Ride-Sharing. ERIM Report Series Reference No. ERS-2010-010-LIS. Available at SSRN: http://ssrn.com/abstract=1568676, 2009.

(5) Herbawi, W., and M. Weber. A genetic and insertion heuristic algorithm for solving the dynamic ridematching problem with time windows. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference* (GECCO '12), Terence Soule (Ed.). ACM, New York, NY, USA, 2012.

(6) Heinrich, S. Implementing Real-time Ridesharing in the San Francisco Bay Area. Masters of Science thesis, San Jose State University, 2010.

Proposition 1. (5.13) registers all drivers who collectively construct each rider's route plan.
Proof. From (5.13) we have $U_r^d \geq X_l^{rd}$. If $X_l^{rd} = 1$, U_r^d is forced to be 1. If $X_l^{rd} = 0$, U_r^d can take either 0 or 1. The term $\sum_{\substack{r\in R_r \\ d\in D_r}} U_r^d$ in the objective function, ensures that U_r^d takes the value of 0.

Proposition 2. Number of connections for rider r can be calculated using term $\sum_{d\in D_r} U_r^d - 1$.
Proof. If driver d carries rider r on any link, then U_r^d=1. So $\sum_{d\in D_r} U_r^d - 1$ can give the number of connections, only if a driver doesn't pick up a rider multiple times. Without loss of generality, we use the example in Figure P.1 to show by contradiction that such a situation cannot happen.

Figure P.1 shows a rider's route plan. On the first and third link, the rider is traveling with d_1, and on the second link, he/she is travelling with d_2. If d_1 travels on both l_1 and l_3, at some point d_1 must have gone from node 2 to node 3, and the rider could have accompanied him on that ride too, reducing the number of connections from 2 to 0. The term $\sum_{\substack{r\in R_r \\ d\in D_r}} U_r^d$ in the minimization objective function ensures that the route with zero connections is selected as the optimal solution.

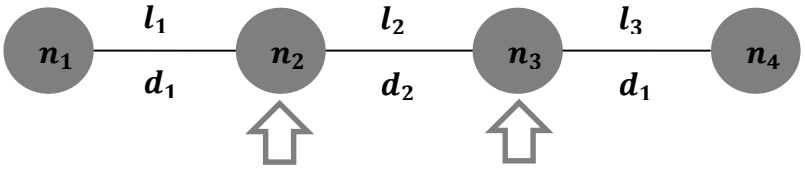


FIGURE P.1 an undesirable match

Proposition 3. Let $X_l^d \in \{0,1\}, U_r^d \in \{0,1\}, X_l^{rd} \in [0\ 1], Y_r \in [0\ 1]$ be the decision variables satisfying the constraint set (5.2)-(5.18), and minimizing the objective function (5.1). All variables have binary values in the optimal solution.
Proof. Let $X_l^d \in \{0,1\}$. Two cases can arise from constraint (5.11), $X_l^{rd} \leq X_l^d$:
Case 1. $X_l^d = 0$. From (5.11), $X_l^{rd} \leq 0$, and hence X_l^{rd} is forced to be zero.
Case 2. $X_l^d = 1$. From (5.11), $X_l^{rd} \leq 1$. Let us assume $X_l^{rd} = f$ where f is a fractional number. Since X_l^{rd} holds a positive value, from (5.7)-(5.9) there exists a route plan or rider r, and link l and driver d construct part of this route plan. Rider's balance constraint (5.8) makes sure that all the X_l^{rd} variables on the rider's path hold a positive value (not necessarily f). (5.7)/(5.9) ensure that Y_r holds a positive value as well. On the other hand, the minimization objective function includes the term Y_r with a negative sign, i.e. the objective function forces the solution to take the highest possible value of Y_r, which is 1.

It is possible, however, for the system of constraints to find multiple paths for one rider, allocating each one a fraction of a trip, such that sum of fractions end up to be 1. Such solutions will not be optimal, because of the term $\sum_{\substack{r\in R_r \\ d\in D_r}} U_r^d$ in the objective function. Multiple paths for a rider translates into strictly higher values for $\sum_{\substack{r\in R_r \\ d\in D_r}} U_r^d$. The same line of reasoning applies to the case of multiple riders, each with a fractional Y_r value.

To conclude, in case there exists at least one route plan for a rider, the problem finds the one with the least number of transfers, and reaches optimality at $Y_r = 1$, and constraints (57)-(5.9) ensure that all X_l^{rd}s on the optimal route plan take the value of 1 as well. In case there is not such a path, these constraints along with (5.11) will set all the value of X_l^{rd} , $\forall d: (r,d) \in M, \forall l \in L_{rd}$ and Y_r to zero.