Algorithms for Trip-Vehicle Assignment in Ride-Sharing

Xiaohui Bei
School of Physical and Mathematical Sciences
Nanyang Technological University
Singapore

Shengyu Zhang
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Hong Kong

Abstract

We investigate the ride-sharing assignment problem from an algorithmic resource allocation point of view. Given a number of requests with source and destination locations, and a number of available car locations, the task is to assign cars to requests with two requests sharing one car. We formulate this as a combinatorial optimization problem, and show that it is NP-hard. We then design an approximation algorithm which guarantees to output a solution with at most 2.5 times the optimal cost. Experiments are conducted showing that our algorithm actually has a much better approximation ratio (around 1.2) on synthetically generated data.

Introduction

The sharing economy is estimated to grow from $14 billion in 2014 to $335 billion by 2025 (Yaraghi and Ravi 2017). As one of the largest components of sharing economy, ride-sharing provides socially efficient transport services that help to save energy and to reduce congestion. Uber has 40 million monthly active riders reported in October 2016 (Kokalitcheva 2016) and Didi Chuxing has more than 400 million users(Tec 2017). A large portion of the revenue of these companies comes from ride sharing with one car catering two passenger requests, which is the topic investigated in this paper. A typical scenario is as follows: There are a large number of requests with pickup and drop-off location information, and a large number of available cars with current location information. One of the tasks is to assign the requests to the cars, with two requests for one car. The assignment needs to be made socially efficient in the sense that the ride sharing does not incur much extra traveling distance for the drivers or and extra waiting time for the passengers.

In this paper we investigate this ride-sharing assignment problem from an algorithmic resource allocation point of view. Formally, suppose that there are a set R of requests $\{(s_i, t_i) \in \mathbb{R}^2 : i = 1, \ldots, m\}$ where in request i, an agent is at location s_i and likes to go to location t_i. There are also a set D of taxis $\{d_k \in \mathbb{R}^2 : k = 1, \ldots, n\}$, with taxi k currently at location d_k. The task is to assign two agents i and j to one taxi k, so that the total driving distance is as small as possible. The distance measure $d(x, y)$ here can be Manhattan distance (i.e., ℓ_1-norm), Euclidean distance (i.e., ℓ_2-norm), or distance on graphs if a city map is available. Here for any fixed tuple $(k, \{i, j\})$, the driver of taxi k has four possible routes, from the combination of the following two choices: he can pick agent i first or agent j first, and he can drop agent i first or drop agent j first. We assume that the driver is experienced enough to take the best among these four choices. Thus we use the total distance of this best route as the driving cost of tuple $(k, \{i, j\})$, denoted by $cost(k, \{i, j\})$. We hope to find an assignment

$$M = \{(k, \{i, j\}) : 1 \le i, j \le m, 1 \le k \le n\}$$

that assigns the maximum number of requests, and in the meanwhile with the $cost(M) = \sum_{(k,\{i,j\}) \in M} cost(k, \{i, j\})$, summation of the driving cost, as small as possible. Here an assignment is a matching in the graph in the sense that each element in $R \cup D$ appears at most once in M.

In this paper, we formulate this ride-sharing assignment as a combinatorial optimization problem. We show that the problem is NP-hard, and then present an approximation algorithm which, on any input, runs in time $O(n^3)$ and outputs a solution M with $cost(M)$ at most 2.5 times the optimal value. Our algorithm does not assume specific distance measure; indeed it works for any distance[1]. We conducted experiments where inputs are generated from uniform distributions and Gaussian mixture distributions. The approximation ratio on these empirical data is about 1.1-1.2, which is much better than the worst case guarantee 2.5. In addition, the results indicate that the larger n and m are, the better the approximation ratio is. Considering that n and m are very large numbers in practice, the performance of our algorithm may be even more satisfactory for practical scenarios.

Related Work

Ridesharing has become a key feature to increase urban transportation sustainability and is an active field of research. Several pieces of work have looked at dynamic ridesharing (Caramia et al. 2002; Fabri and Recht 2006; Agatz et al. 2012; Santos and Xavier 2013; Alonso-Mora et al. 2017), and multi-hop ridesharing (Herbawi and Weber 2011; Drews and Luxen 2013; Teubner and Flath 2015).

[1]That is, the algorithm only needs that d is nonnegative, symmetric and satisfies the triangle inequality.

Another closely related optimization problem is known as the dial-a-ride problem (Cordeau 2006), where the task is to transport people between pickup and delivery locations with different transportation constraints, such as time window and maximum ride time limits. Many variants of the dial-a-ride problem were proposed depending on the specific applications. See (Cordeau and Laporte 2007; Parragh, Doerner, and Hartl 2010) for an overview.

Our model is also closely related to the three-dimensional assignment problem(3DA) (Crama and Spieksma 1992; 1992; Pferschy, Rudolf, and Woeginger 1994). Given three disjoint sets of points, each with size n, the problem asks to find a minimum-weight collection of n triangles covering each point exactly once, where the weight of a triangle is defined as either the sum of lengths of its sides or the sum of the length of the two shortest sides. There are two main differences between the 3DA problem and our model: (1) in the 3DA problem every triangle must contain one point from each set, while in our model every driver-requests matching contains one driver and two ride requests; (2) we further generalize the 3DM problem by considering a pickup and a drop off location for each ride request.

Mechanisms for ridesharing have also been studied (Kamar and Horvitz 2009; Kleiner, Nebel, and Ziparo 2011; Shen, Lopes, and Crandall 2016), where the goal is to design incentive compatible mechanisms that provide fair and efficient assignment solutions.

Besides ridesharing, the problem of allocating shareable resources has also been studied in other applications, such as high dimensional stable matching (Boros et al. 2004; Eriksson, Sjöstrand, and Strimling 2006; Huang 2007), and the roommate assignment problem (Abdulkadiroğlu, Sönmez, and Ünver 2004; Chan et al. 2016; Huzhang et al. 2017).

Preliminaries

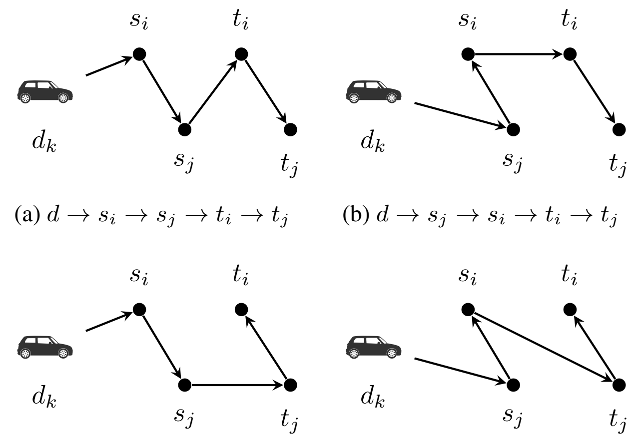
For an positive integer n, we adopt the notation $[n] \stackrel{\text{def}}{=} \{1, \ldots, n\}$. For a set S, the set of unordered pairs of elements from S is denoted by $\binom{S}{2}$.

A weighted graph G is a tuple (V, E, w), where V is the vertex set, E is the edge set and $w : E \to \mathbb{R}$ is the edge weight function. We often drop E when $E = \binom{V}{2}$. In this paper the weight w can be any metric satisfying non-negativity ($w(x, y) \geq 0$), symmetry ($w(x, y) = w(y, x)$) and the triangle inequality ($w(x, y) + w(y, z) \geq w(x, z)$). In practice, the distance function can be ℓ_2, ℓ_1, or distance on a road graph. We extend the weight notation to paths:

$$w(a_1, a_2, \ldots, a_k) = \sum_{i=1}^{k-1} w(a_i, a_{i+1}).$$

A perfect matching M in a graph is a maximal set of vertex-disjoint edges. A minimum matching M in a weighted graph is a perfect matching with minimum total weight. A minimum matching M in a weighted graph of n vertices can be found in time $O(n^3)$ (say, by reducing it to a maximum weight matching) (Gabow 1990).

Problem formulation. Suppose that we have a set of m requests $R = \{1, \ldots, m\}$, where each request i contains a pickup location s_i and a drop-off location t_i. We also have a set of n available drivers $D = \{1, \ldots, n\}$, with driver k at location d_k. We would like to pair up requests and also match drivers to the paired requests. More precisely, we aim to find an allocation $M = \{(k, R_k) : k \in D\}$, where $R_k \subseteq R$ and $R_1, R_2, \ldots R_n$ are mutually disjoint subsets of requests. For each driver $k \in D$, this allocation assigns k to serve all requests in R_k. Given the low-capacity nature of ride-sharing applications, in this paper we restrict ourselves to the case with $|R_k| \leq 2$ for every k.



(a) $d \to s_i \to s_j \to t_i \to t_j$ (b) $d \to s_j \to s_i \to t_i \to t_j$

(c) $d \to s_i \to s_j \to t_j \to t_i$ (d) $d \to s_j \to s_i \to t_j \to t_i$

Figure 1: Four possible routes for driver k to serve requests i and j.

Next we define a cost function $\mathit{cost}(k, R_k)$ for single-driver routing. This function returns the distance of the shortest distance that driver k needs to travel to serve R_k. It is easy to see that when R_k has only one request i, $\mathit{cost}(k, R_k) = w(d_k, s_i) + w(s_i, t_i)$. When R_k has two requests i and j, there are four routes one can choose, corresponding to different orders to pick up and drop off the two passengers (Figure 1). Thus

$$\begin{aligned} \mathit{cost}(k, \{i, j\}) = \min\{ & w(d_k, s_i, s_j, t_i, t_j), \\ & w(d_k, s_i, s_j, t_j, t_i), \\ & w(d_k, s_j, s_i, t_i, t_j), \\ & w(d_k, s_j, s_i, t_j, t_i)\} \end{aligned}$$

The cost of an allocation M is in turn defined as

$$\mathit{cost}(M) = \sum_{(k, R_k) \in M} \mathit{cost}(k, R_k).$$

Given input weighted graph $G = (V, E, w)$, the pickup and drop-off locations $L_R = \{(s_i, t_i) \in V^2 : i \in R\}$ and the driver locations $L_D = \{d_k \in V : d \in D\}$, the *ride-sharing problem* asks to find an allocation that assigns the maximum number of requests with the minimum total cost.

Remark. When defining the possible routes for driver k to pick up requests i and j, there are two additional routes that can be taken into consideration:

$$(d_k \to s_i \to t_i \to s_j \to t_j) \quad \text{and}$$
$$(d_k \to s_j \to t_j \to s_i \to t_i).$$

Note that both of these routes complete one request before serving the other, hence they do not fit into the "ridesharing" category in a strict sense. Nevertheless, our results can be easily generalized to include these two possibilities. The only changes are to add these two routes to the definition of *cost* function, and to Step (2) and (3) of **Algorithm 1** described later. All proofs and analysis remain unchanged.

NP Hardness

In this section we will show that the ride-sharing problem defined in the last section is NP-hard.

Theorem 1 *The ride-sharing problem is NP-hard.*

Proof We show a reduction from the 3-dimensional perfect matching problem (3DM), which is known to be NP-hard (Garey and Johnson 1990). Recall that 3DM is as follows: Given three finite and disjoint sets I, J, K, each of size n, and a subset $T \subseteq I \times J \times K$ with size $m \geq n$ (i.e. T consists of triples (i, j, k) such that $i \in I, j \in J$, and $k \in K$), the 3DM problem asks if there exists a subset $M \subseteq T$ with n triples, such that every element in $I \cup J \cup K$ occurs in exactly one triple of M.

Consider a 3DM problem instance $\mathcal{I} = \{I_0, J_0, K_0, T_0\}$. We construct a ride-sharing problem instance as follows. There are $n + 3m$ drivers

$$D = K_0 \cup \{k_i(e), k_j(e), k_r(e) : e \in T_0\},$$

where $k_i(e), k_j(e), k_r(e)$ are (the IDs of) three new drivers introduced with each edge e, and $2n + 6m$ requests

$$\begin{aligned} R = & I_0 \cup \{i_0(e), i_1(e) : e \in T_0\} \cup \\ & J_0 \cup \{j_0(e), j_1(e) : e \in T_0\} \cup \\ & \{r_0(e), r_1(e) : e \in T_0\}. \end{aligned}$$

where $i_0(e), i_1(e), j_0(e), j_1(e), r_0(e), r_1(e)$ are (the IDs of) the new requests introduced for each edge e.

For each request $i \in R$, we assume that $s_i = t_i$, and will show that even for this degenerate case, the allocation problem is still NP-hard. Note that now we have

$$\begin{aligned} cost(k, \{i, j\}) = \min\{ & w(d_k, s_i) + w(s_i, s_j), \\ & w(d_k, s_j) + w(s_j, s_i)\}. \end{aligned}$$

The distance function w are defined by a graph G that contains all locations of drivers and requests as vertices. More specifically, G consists of m subgraphs $G(e)$, indexed by elements $e = (i, j, k) \in T_0$. Each $G(e)$ is represented in Figure 2. For any two locations $\ell_1, \ell_2 \in G$, we let $w(\ell_1, \ell_2) = 1$ if (ℓ_1, ℓ_2) is an edge in G, and $w(\ell_1, \ell_2) = 2$ otherwise. It is easily seen that the triangle inequality is satisfied for this weight function.

Given this construction, we will next show that the 3DM problem has perfect matching if and only if the ride-sharing problem has an optimal allocation with total cost $2n + 6m$. First, if the 3DM problem has a perfect matching M, then

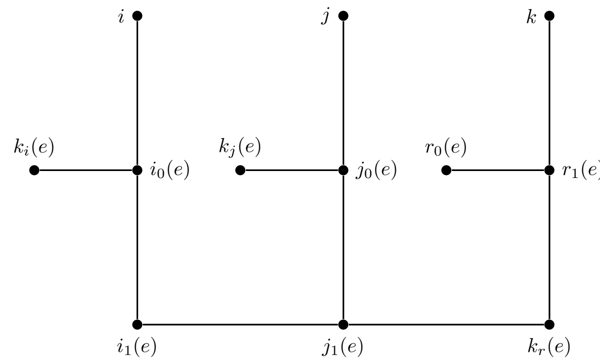


Figure 2: Component G_e.

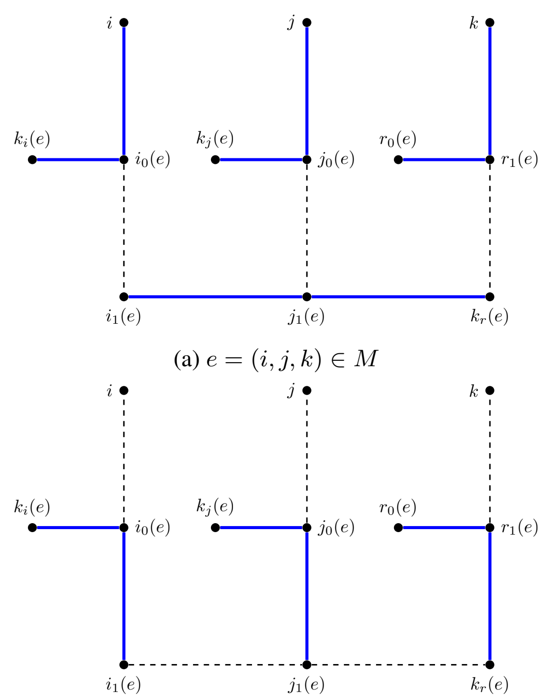
there is an optimal allocation for the ride-sharing problem consisting of

$$\begin{aligned} \{(k_r(e), \{i_1(e), j_1(e)\}), \quad & (k_i(e), \{i_0(e), i\}), \\ (k_j(e), \{j_0(e), j\}), \quad & (k, \{r_0(e), r_1(e)\})\} \end{aligned}$$

for each $e = (i, j, k) \in M$ (demonstrated in Figure 3a), and

$$\begin{aligned} & \{(k_i(e), \{i_0(e), i_1(e)\}), \quad (k_j(e), \{j_0(e), j_1(e)\}), \\ & (k_r(e), \{r_0(e), r_1(e)\})\} \end{aligned}$$

for each $e = (i, j, k) \in T_0 \backslash M$ (demonstrated in Figure 3b). The total cost of this allocation is $2n + 6m$, which is the smallest over all possible allocations.



(a) $e = (i, j, k) \in M$

(b) $e = (i, j, k) \in T_0 \backslash M$

Figure 3

On the other hand, if the ride-sharing problem admits an allocation of total cost $2n + 6m$, it is not hard to see that the allocation must be of the above format, from which we can derive a perfect matching for the 3DM problem. This completes the NP-hardness proof. $\square$

Algorithm

In this section, we present an approximation algorithm for the ride sharing problem. For simplicity, we assume that $m = 2n$ at this stage.

The algorithm takes a two-phase greedy approach. In the first phase, it matches the $2n$ requests into n pairs based on the shortest distance to serve any request pair but on the worse pickup choice. In the second phase, we assign drivers to the pairs formed in the previous phase, under the assumption that the distance from a driver k to a pair of requests is distance from d_k to the nearer pickup location of the two.

The algorithm is given in **Algorithm 1**. Recall that $R = \{1, \ldots, m\}$, $D = \{1, \ldots, n\}$, and that the input consists of the weighted graph $G = (V, E, w)$, the pickup and drop-off locations $L_R = \{(s_i, t_i) \in V^2 : i \in R\}$ and the driver locations $L_D = \{d_k \in V : d \in D\}$.

Algorithm 1 Allocation(G, L_R, L_D)

Input: A nonnegative weighted graph $G = (V, E, w)$, request locations $L_R = \{(s_i, t_i) \in V^2 : i \in R\}$, driver locations $L_D = \{d_k \in V : d \in D\}$.
Output: An allocation $M = \{(k, \{i, j\}) : k \in D, i, j \in R\}$.

for $i, j \in R$ **do**
  $u_{ij} = \min\{w(s_i, s_j, t_i, t_j), w(s_i, s_j, t_j, t_i)\}$
  // the shortest route that picks up request i first
  $u_{ji} = \min\{w(s_j, s_i, t_i, t_j), w(s_j, s_i, t_j, t_i)\}$
  // the shortest route that picks up request j first
  $v^1(\{i, j\}) = \max\{u_{ij}, u_{ji}\}$.
end for
Find a minimum weight perfect matching M_1 in the weighted graph $G_1 \stackrel{\text{def}}{=} (R, v^1)$.
for $k \in D$ and $\{i, j\} \in M_1$ **do**
  $v^2(k, \{i, j\}) = \min\{w(d_k, s_i), w(d_k, s_j)\}$.
end for
Find a minimum weight perfect matching M_2 in the weighted bipartite graph $G_2 \stackrel{\text{def}}{=} (D, M_1, v^2)$.
Output $M = M_2$

Analysis

Theorem 2 *On any input, Algorithm 1 runs in time $O(n^3)$ and outputs a solution M with $cost(M)$ at most 2.5 times the optimal value.*

To prove Theorem 2, we further introduce the following notation. Fix an optimal solution $M^* = \{(k, R_k = \{i, j\}) \mid$ the optimal solution assigns driver k to pick up i and $j\}$, and let $M_R^* = \{R_k \mid (k, R_k) \in M^*\}$.

For every $k \in D$ and $R_k \subseteq R$, recall that $cost(k, R_k)$ is the shortest distance for driver k to serve requests in R_k. Note that $cost(k, R_k)$ consists of two parts. We let

$$cost(k, R_k) = cost_D(k, R_k) + cost_R(k, R_k),$$

where $cost_D(k, R_k)$ is the distance from d_k to the first pickup location in the optimal route, and $cost_R(k, R_k)$ is the distance from the first pickup location to the last drop-off in the optimal route. Given an allocation M, we then define $cost_R(M) = \sum_{(k,R_k)\in M} cost_R(k, R_k)$.

The following two lemmas relate the total costs of M_1 and M_2 (in the algorithm) to their corresponding parts in M^*. By slight abuse of notation, in the following we use $v^1(M_1)$ to denote $\sum_{R_k \in M_1} v^1(R_k)$ and $v^2(M_2)$ to denote $\sum_{(k,R_k)\in M_2} v^2(k, R_k)$.

Lemma 3 $v^1(M_1) \leq cost_R(M^*) + \sum_{\{i,j\}\in M_R^*} w(s_i, s_j)$

Proof Consider any $(k, R_k = \{i, j\}) \in M^*$. Assume without loss of generality that in the optimal solution, driver d_k picks up passenger i at s_i first. That is, $cost_R(k, R_k) = u_{ij} = \min\{w(s_i, s_j, t_i, t_j), w(s_i, s_j, t_j, t_i), w(s_i, t_i, s_j, t_j)\}$. Consider two possibilities:

- $v^1(\{i, j\}) = u_{ij} = cost_R(k, \{i, j\})$;
- $v^1(\{i, j\}) = u_{ji}$. By triangle inequality, it is not hard to see that $v^1(\{i, j\}) \leq cost_R(k, \{i, j\}) + w(s_i, s_j)$.

Adding above (in)equalities together for every $(k, R_k) \in M^*$ gives $v^1(M_R^*) \leq cost_R(M^*) + \sum_{\{i,j\}\in M_R^*} w(s_i, s_j)$. Next note that M_1 is the minimum weight matching with regard to v^1, hence

$$v^1(M_1) \leq v^1(M_R^*) \leq cost_R(M^*) + \sum_{\{i,j\}\in M_R^*} w(s_i, s_j).$$

$\square$

Lemma 4 $v^2(M_2) \leq \frac{1}{2} \sum_{(k,\{i,j\})\in M^*} (w(d_k, s_i) + w(d_k, s_j))$.

Proof Consider graph

$$G' = (V, M_1 \cup \{(\{i, k\}, \{j, k\}) : (k, \{i, j\}) \in M^*\}).$$

Note that every vertex has degree 2 in graph G'. Thus G' consists of a collection of disjoint cycles. Pick an arbitrary such cycle C, C can be written as $(i_1, j_1, k_1, i_2, j_2, k_2, \ldots, i_t, j_t, k_t, i_1)$, where $\{i_a, j_a\} \in M_1$ and $k_a \in D$ for each $1 \leq a \leq t$. Thus C can be partitioned into 3 matchings

- $M_1^C = M_1 \cap C$
- $M_2^C = \{\{j_1, k_1\}, \{j_2, k_2\}, \ldots, \{j_t, k_t\}\}$
- $M_3^C = \{\{k_1, i_2\}, \{k_2, i_3\}, \ldots, \{k_t, i_1\}\}$.

Combining all cycles together, we have $M_1 = \bigcup_{C\in G'} M_1^C$. Let $M_2^{G'} = \bigcup_{C\in G'} M_2^C$ and $M_3^{G'} = \bigcup_{C\in G'} M_3^C$, then both are perfect matchings in $G_2 = (D, M_1, v^2)$. Recall that M_2 is a minimum weight perfect matching in $G_2 = (D, M_1, v^2)$, and v^2 is so defined that

$$v^2(k, \{i, j\}) \leq w(d_k, s_i), \text{ and } v^2(k, \{i, j\}) \leq w(d_k, s_j).$$

Therefore, we have both

$$v^2(M_2) \leq \sum_{\{j,k\} \in M_2^{G'}} w(d_k, s_j)$$

and

$$v^2(M_2) \leq \sum_{\{k,i\} \in M_3^{G'}} w(d_k, s_i).$$

Thus

$$\begin{aligned} v^2(M_2) &\leq \frac{1}{2} \left(\sum_{\{j,k\} \in M_2^{G'}} w(d_k, s_j) + \sum_{\{k,i\} \in M_3^{G'}} w(d_k, s_i) \right) \\ &= \frac{1}{2} \sum_{(k,\{i,j\}) \in M^*} (w(d_k, s_i) + w(d_k, s_j)). \end{aligned}$$

□

Lemma 5 *The total cost of M^* satisfies*

$$cost(M^*) \geq \frac{1}{2} \sum_{(k,\{i,j\}) \in M^*} (w(s_i, s_j) + w(d_k, s_i) + w(d_k, s_j))$$

Proof Suppose that for driver k to serve requests i and j, the best route is to first pick up i (the other case is symmetric). Then

$$\begin{aligned} cost(k, \{i, j\}) &= w(d_k, s_i) + u_{ij} \\ &\geq w(d_k, s_i) + w(s_i, s_j). \end{aligned} \tag{1}$$

By triangle inequality, we have $w(d_k, s_i) \geq w(d_k, s_j) - w(s_i, s_j)$. Plugging this into Eq.(1) gives

$$cost(k, \{i, j\}) \geq w(d_k, s_j). \tag{2}$$

Adding Eq.(1) and Eq.(2) and dividing by 2 proves the lemma as desired. □

Now we are ready to prove Theorem 2.

Proof [of Theorem 2]

Let M be the allocation returned by the algorithm. We have

$$\begin{aligned} & cost(M) \\ = & \sum_{(k,\{i,j\}) \in M} \min\{w(d_k, s_i) + u_{ij}, w(d_k, s_j) + u_{ji}\} \\ \leq & \sum_{(k,\{i,j\}) \in M} \min\{w(d_k, s_i), w(d_k, s_j)\} + v^1(\{i, j\}) \\ = & v^1(M_1) + \sum_{(k,\{i,j\}) \in M} \min\{w(d_k, s_i), w(d_k, s_j)\} \\ = & v^1(M_1) + v^2(M_2) \end{aligned}$$

Next we plug in Lemma 3 and Lemma 4 to replace $v^1(M_1)$ and $v^2(M_2)$. This gives us

$$\begin{aligned} & v^1(M_1) + v^2(M_2) \\ \leq & cost_R(M^*) + \sum_{\{i,j\} \in M_R^*} w(s_i, s_j) \\ & + \frac{1}{2} \sum_{(k,\{i,j\}) \in M^*} (w(d_k, s_i) + w(d_k, s_j)) \\ = & cost_R(M^*) + \frac{1}{2} \sum_{\{i,j\} \in M_R^*} w(s_i, s_j) \\ & + \frac{1}{2} \sum_{(k,\{i,j\}) \in M^*} (w(s_i, s_j) + w(d_k, s_i) + w(d_k, s_j)) \\ \leq & cost_R(M^*) + \frac{1}{2} \sum_{\{i,j\} \in M_R^*} w(s_i, s_j) + cost(M^*) \\ & \text{(by Lemma 5)} \\ \leq & cost(M^*) + \frac{1}{2} cost(M^*) + cost(M^*) \\ = & \frac{5}{2} cost(M^*) \end{aligned}$$

□

Remark. In this algorithm we assume $m = 2n$, i.e., the number of requests is exactly twice the number of drivers. A natural open question is to relax this assumption and consider more general conditions. When $m < 2n$, the solution would allow some drivers to serve only one or even zero request. When $m > 2n$, the problem becomes to find $2n$ requests among the m requests for the drivers to serve with minimum total cost. We conjecture that some natural variants of **Algorithm 1** could solve these problems with similar approximation guarantees. The performance of one such variant to handle the case with $m > 2n$ is demonstrated in the following section.

Experiments

We conducted several experiments to show that empirically our algorithm has a better approximation ratio than the theoretic guarantee of 2.5 in the worst case in the last section. We consider region $C \stackrel{\text{def}}{=} [0, B] \times [0, B]$ for a big number B (such as 100). We tested two distance measures: ℓ_1-norm and ℓ_2-norm. The inputs $\{s_i, t_i, d_k : i \in R, k \in D\}$ in two ways. In the first, we generate all s_i, t_i, d_k independently from uniform distribution on C. The second distribution for inputs s_i, t_i, d_k is Gaussian mixture, where we have a number of centers $\{\mu_1, \ldots, \mu_c\}$ and covariance matrices $\Sigma_1, \ldots, \Sigma_c$. Each $s_i = (x_i^s, y_i^s)$ is drawn in a two-step procedure: First sample a $j \in [c]$ from a probability distribution p over $[c]$, and then sample s_i from the two-dimensional Gaussian $N(\mu_j, \Sigma_j)$. For simplicity, we take p to be uniform and $\Sigma_j = \begin{pmatrix} \sigma, 0 \\ 0, \sigma \end{pmatrix}$ for some positive parameter σ.

	$n=10$	$n=20$	$n=30$	$n=40$	$n=50$
B=10	ℓ_1: 1.23	ℓ_1: 1.18	ℓ_1: 1.16	ℓ_1: 1.14	ℓ_1: 1.10
	ℓ_2: 1.21	ℓ_2: 1.20	ℓ_2: 1.15	ℓ_2: 1.14	ℓ_2: 1.12
B=50	ℓ_1: 1.23	ℓ_1: 1.19	ℓ_1: 1.17	ℓ_1: 1.16	ℓ_1: 1.15
	ℓ_2: 1.19	ℓ_2: 1.18	ℓ_2: 1.18	ℓ_2: 1.17	ℓ_2: 1.16
B=100	ℓ_1: 1.24	ℓ_1: 1.21	ℓ_1: 1.20	ℓ_1: 1.21	ℓ_1: 1.18
	ℓ_2: 1.21	ℓ_2: 1.22	ℓ_2: 1.18	ℓ_2: 1.16	ℓ_2: 1.16

Table 1: Approximation ratios for uniform distribution

	1 center	5 centers	10 centers
$\sigma=1$	ℓ_1: 1.19	ℓ_1: 1.14	ℓ_1: 1.15
	ℓ_2: 1.17	ℓ_2: 1.16	ℓ_2: 1.15
$\sigma=5$	ℓ_1: 1.16	ℓ_1: 1.14	ℓ_1: 1.16
	ℓ_2: 1.16	ℓ_2: 1.13	ℓ_2: 1.17
$\sigma=10$	ℓ_1: 1.15	ℓ_1: 1.14	ℓ_1: 1.17
	ℓ_2: 1.18	ℓ_2: 1.14	ℓ_2: 1.17

Table 2: Approximation ratios for Guassian mixture distribution

As finding the exact optimal value OPT is NP hard, we can only compare the output of our algorithm with a lower bound L of the optimal value. The lower bound we use is the following. Take a minimum weight perfect matching M_1' in graph $G_1 = ([m], cost_{\min})$ where $cost_{\min}(i,j) = \min\{u_{ij}, u_{ji}\}$. Then take a minimum weight bipartite perfect matching M_2' in $G_2 = (R, D, w)$. It is not hard to see that $L \stackrel{\text{def}}{=} cost_{\min}(M_1') + w(M_2')$ is a lower bound of the optimal value OPT. Indeed, as the optimal solution can be partitioned into two parts, M_1^* and M_2^*, where M_1^* is a perfect matching in G_1 and M_2^* is a bipartite perfect matching in G_2. As M_1' and M_2' are the minimum weight perfect matchings in G_1 and G_2 respectively, so

$$\begin{aligned} L &= cost_{\min}(M_1') + w(M_2') \\ &\leq cost_{\min}(M_1^*) + w(M_2^*) \\ &\leq cost(M_1^*) + w(M_2^*) \\ &\leq OPT, \end{aligned}$$

and therefore

$$\alpha' \stackrel{\text{def}}{=} \frac{cost(M)}{L} \quad \geq \quad \alpha \stackrel{\text{def}}{=} \frac{cost(M)}{OPT}. \tag{3}$$

In other words, for any output allocation M, the quantity α' defined as $cost(M)/L$ is an upper bound of the true approximation ratio α of this solution M. And we will show that even this α' is empirically very small.

The results for inputs drawn from the uniform distribution are illustrated in Table 1, with different parameter values of B, n tested. The results for Gaussian mixture are illustrated in Table 2, with different parameter values of σ, c tested (and B fixed to 100, n fixed to 50).

From these results, we can observe the following.

1. Though the theoretic upper bound is 2.5 as shown in the previous section, even the upper bound α' of the approximation ratio is only about 1.1-1.2 on empirical data.

	1 center	5 centers	10 centers
$\sigma=1$	ℓ_1: 1.91	ℓ_1: 1.95	ℓ_1: 1.89
	ℓ_2: 1.91	ℓ_2: 1.86	ℓ_2: 1.96
$\sigma=5$	ℓ_1: 1.89	ℓ_1: 1.88	ℓ_1: 1.89
	ℓ_2: 1.92	ℓ_2: 1.94	ℓ_2: 1.87
$\sigma=10$	ℓ_1: 1.87	ℓ_1: 1.89	ℓ_1: 1.91
	ℓ_2: 1.85	ℓ_2: 1.88	ℓ_2: 1.93

Table 3: Approximation ratios for Guassian mixture distribution when $m = 3n$

2. The empirical approximation ratio upper bound α' varies little with parameters B, c, σ and choice of norms (ℓ_1 or ℓ_2), but it does decrease with n. Considering that in practice the number of requests and the number of taxis are huge, our algorithm likely exhibits even better approximation ratio.

We also demonstrate the performance of a variant of Algorithm 1 for the case $m > 2n$, i.e., the number of requests is more than twice the number of drivers. Our goal is to assign $2n$ out of the m requests to the n drivers to serve with minimum total cost. To handle this case, we change Step (6) of Algorithm 1 to "Find a minimum weight matching of size n". The rest parts of Algorithm 1 remain the same. We test this algorithm on inputs drawn from a Gaussian mixture distribution. All parameters are the same as in Table 2, except that we set $n = 50$ and $m = 150$. The results are illustrated in Table 3. We observe that the upper bound for the approximation ratio drops to around 1.8-1.9 for most cases. Note that the upper bound itself is worse than the case of $m = 2n$, because now with more requests, the minimum weight matching have a lower cost. Therefore, the apparently worse ratio of 1.8-1.9 does not necessarily mean that the algorithm is bad in the case of $m > 2n$; it may be our analysis is looser. How to further improve the analysis, the performance of these algorithms, and prove that they deliver similar approximation guarantees remain as interesting open questions for future studies.

Concluding Remarks and Open Questions

In this work, we formulate the ride-sharing as a combinatorial optimization problem, and show that it is NP-hard. We design an approximation algorithm which guarantees to output a solution with at most 2.5 times the optimal cost, and the algorithms runs in polynomial time. Experiments are conducted showing that our algorithm actually has a much better approximation ratio (around 1.1-1.2) on synthetically generated data, and the ratio decreases as m and n increase.

Our results lead to many future working directions. One open question, as we discussed earlier, is to relax the $m = 2n$ assumption and consider more general conditions. Another interesting direction is to add time constraints to the model and study the dynamic real-time assignment problem.

Acknowledgment S.Z. is partially supported by Research Grants Council of the Hong Kong S.A.R. (Project no.

CUHK14239416). We thank Zhiyi Huang and Xin Huang for inspiring discussions at the early stage of the research.

References

Abdulkadiroğlu, A.; Sönmez, T.; and Ünver, M. U. 2004. Room assignment-rent division: A market approach. *Social Choice and Welfare* 22(3):515–538.

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

Alonso-Mora, J.; Samaranayake, S.; Wallar, A.; Frazzoli, E.; and Rus, D. 2017. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences* 201611675.

Boros, E.; Gurvich, V.; Jaslar, S.; and Krasner, D. 2004. Stable matchings in three-sided systems with cyclic preferences. *Discrete Mathematics* 289(1):1–10.

Caramia, M.; Italiano, G. F.; Oriolo, G.; Pacifici, A.; and Perugia, A. 2002. Routing a fleet of vehicles for dynamic combined pick-up and deliveries services. In *Operations Research Proceedings 2001*, 3–8. Springer.

Chan, P. H.; Huang, X.; Liu, Z.; Zhang, C.; and Zhang, S. 2016. Assignment and pricing in roommate market. In *AAAI*, 446–452.

Cordeau, J.-F., and Laporte, G. 2007. The dial-a-ride problem: models and algorithms. *Annals of operations research* 153(1):29.

Cordeau, J.-F. 2006. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research* 54(3):573–586.

Crama, Y., and Spieksma, F. C. 1992. Approximation algorithms for three-dimensional assignment problems with triangle inequalities. *European Journal of Operational Research* 60(3):273–279.

Drews, F., and Luxen, D. 2013. Multi-hop ride sharing. In *Sixth annual symposium on combinatorial search*.

Eriksson, K.; Sjöstrand, J.; and Strimling, P. 2006. Three-dimensional stable matching with cyclic preferences. *Mathematical Social Sciences* 52(1):77–87.

Fabri, A., and Recht, P. 2006. On dynamic pickup and delivery vehicle routing with several time windows and waiting times. *Transportation Research Part B: Methodological* 40(4):335–350.

Gabow, H. N. 1990. Data structures for weighted matching and nearest common ancestors with linking. In *SODA*, 434–443.

Garey, M. R., and Johnson, D. S. 1990. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co.

Herbawi, W., and Weber, M. 2011. Evolutionary multiobjective route planning in dynamic multi-hop ridesharing. In *EvoCOP*, volume 11, 84–95. Springer.

Huang, C.-C. 2007. Two's company, three's a crowd: Stable family and threesome roommates problems. *Algorithms–ESA 2007* 558–569.

Huzhang, G.; Huang, X.; Zhang, S.; and Bei, X. 2017. Online roommate allocation problem. In *IJCAI*, 235–241.

Kamar, E., and Horvitz, E. 2009. Collaboration and shared plans in the open world: Studies of ridesharing. In *IJCAI*, volume 9, 187.

Kleiner, A.; Nebel, B.; and Ziparo, V. A. 2011. A mechanism for dynamic ride sharing based on parallel auctions. In *IJCAI*, volume 11, 266–272.

Kokalitcheva, K. 2016. Uber now has 40 million monthly riders worldwide. *Fortune Magazine*.

Parragh, S. N.; Doerner, K. F.; and Hartl, R. F. 2010. Demand responsive transportation. *Wiley Encyclopedia of Operations Research and Management Science*.

Pferschy, U.; Rudolf, R.; and Woeginger, G. J. 1994. Some geometric clustering problems. *Nord. J. Comput.* 1(2):246–263.

Santos, D. O., and Xavier, E. C. 2013. Dynamic taxi and ridesharing: A framework and heuristics for the optimization problem. In *IJCAI*, volume 13, 2885–2891.

Shen, W.; Lopes, C. V.; and Crandall, J. W. 2016. An online mechanism for ridesharing in autonomous mobility-on-demand systems. *arXiv preprint arXiv:1603.02208*.

2017. Didi accelerate globalization with investment in brazilian uber rival 99. *TechNode*. http://technode.com/2017/01/05/didi-accelerates-globalization-with-investment-in-brazilian-uber-rival-99/.

Teubner, T., and Flath, C. M. 2015. The economics of multi-hop ride sharing. *Business & Information Systems Engineering* 57(5):311–324.

Yaraghi, N., and Ravi, S. 2017. The current and future state of the sharing economy. *Brookings India* 032017.