2017 IEEE 37th International Conference on Distributed Computing Systems

Online to Offline Business: Urban Taxi Dispatching with Passenger-Driver Matching Stability

Huanyang Zheng and Jie Wu
Department of Computer and Information Sciences, Temple University, USA
Email: {huanyang.zheng, jiewu}@temple.edu

***Abstract*—In the Online to Offline (O2O) taxi business (e.g., Uber), the interests of passengers, taxi drivers, and the company may not align with one another, since taxis do not belong to the company. To balance these interests, this paper studies the taxi dispatch problem for the O2O taxi business. The interests of passengers and taxi drivers are modeled. For non-sharing taxi dispatches (multiple passenger requests cannot share a taxi), a stable marriage approach is proposed. It can deal with unequal numbers of passenger requests and taxis through matching them to dummy partners. Given dummy partners, stable matchings are proved to exist. Three rules are presented to find out all possible stable matchings. For sharing taxi dispatches (multiple passenger requests can share a taxi), passenger requests are packed through solving a maximum set packing problem. Packed passenger requests are regarded as a single request for matching taxis. Extensive real data-driven experiments demonstrate how well our approach performs. The proposed algorithms have a limited performance gap to the literature in terms of the dispatch delay and the passenger satisfaction, but they significantly improve upon existing algorithms in terms of the taxi satisfaction.**

***Keywords*-Taxi dispatch schedule, passenger requests, taxi drivers, matching stability, sharing and non-sharing.**

I. Introduction

Taxis are playing an increasingly important role in modern cities to support people's daily commutes. Based on a survey conducted in New York city [1], more than 100 taxi companies are operating more than 13,000 taxis in New York, delivering 660,000 passengers every day. They convey more than 25% of all the passengers, accounting for 45% of total transit fares. In traditional taxi companies, taxis and their drivers belong to a company. As a result, taxis are owned by the company, and are dispatched based on the interest of the company (rather than the interest of each taxi driver). Although taxi drivers are coordinated to maximize the total profit of the company, their own profits are not considered in taxi dispatch schedules.

Recently, the taxi business has been revolutionized by the Online to Offline (O2O) business, which combines offline business services with online platforms. Customers are attracted online, but are served offline. One example of an O2O taxi company is Uber Technologies, which creates a mobile application to connect drivers and passengers. The O2O taxi business is different from traditional services, since taxis do not belong to the company. Note that Uber taxis are completely private. Therefore, the interests of Uber drivers may not align with the interest of Uber. Consequently, it becomes extremely challenging to balance the interests of passengers, drivers, and the company in the O2O taxi business.

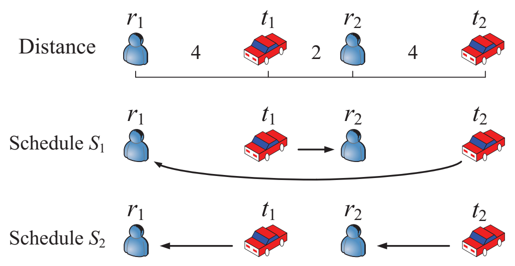


Fig. 1. An illustration of the O2O taxi dispatch.

To better illustrate the taxi dispatch challenge, an example is shown in Fig. 1, which involves two passenger requests (r_1 and r_2) and two taxis (t_1 and t_2). The distances between passengers and taxis have been shown. In the schedule S_1, t_1 is assigned to r_2, and t_2 is assigned to r_1. t_1 and r_2 get their best partners, while t_2 and r_1 get their worst partners. To pick up passengers, the total taxi travel distance in S_1 is 12. In the schedule S_2, t_1 is assigned to r_1, and t_2 is assigned to r_2. r_1 and t_2 get their best partners, while r_2 and t_1 get their worst partners. To pick up passengers, the total taxi travel distance in S_2 is 8. In terms of the total taxi travel distance, S_2 is better than S_1. However, in terms of passenger-driver fairness, S_1 and S_2 are the same (one passenger and one taxi get their best partners, while the others get their worst partners). Since taxis are private and passengers are independent, determining which taxis and passengers should be prioritized is difficult. To determine a taxi dispatch schedule for the O2O taxi business, the balanced interests of passengers, taxi drivers, and the company must be considered.

This paper studies the O2O taxi dispatch problem through a stable marriage approach [2]. Given a set of passenger requests and a set of taxis, the objective is to maximally and stably match passenger requests and taxis. Each passenger always prefers a shorter wait time, and thus, prefers taxis that are closer. On the other hand, the taxi's interest depends on the distance to pick up the passenger, as well as the distance to carry the passenger. The passenger-driver fairness is represented by the matching stability, i.e., a matched passenger and a matched driver will not prefer each other over their existing partners. Our approach can deal with unequal numbers of passenger requests and taxis, while a passenger request or a taxi can be matched to a dummy partner. Our approach is further extended to taxi sharing scenarios, in which multiple passenger requests can share one taxi by solving a maximum set packing problem.

Our main contributions are summarized as follows:

- We discuss novel taxi dispatch issues for the O2O taxi business, in which taxis do not belong to the company. The interests of passengers, drivers, and the company are modeled and balanced.
- We propose a stable marriage approach to dispatch taxis in the O2O taxi business. It can deal with scenarios that have unequal numbers of passenger requests and taxis.
- We investigate taxi sharing problems, in which multiple passenger requests can share one taxi. Passenger requests are packed by solving a maximum set packing problem.
- Extensive real data-driven experiments are conducted to evaluate the proposed solutions. The results are shown from different perspectives to provide conclusions.

The remainder of this paper is organized as follows. Section II surveys related works. Section III describes the model, and then, formulates the problem. Sections IV and V will explore non-sharing and sharing taxi dispatches, respectively. Section VI includes experiments. Section VII concludes the paper.

II. Related Work

In traditional taxi companies, taxis are owned by the company. Consequently, taxis are dispatched based on the interest of the company rather than the interest of each taxi driver. For example, Hanna et al. [3] considered the taxi dispatching issue as an assignment problem, and solved it via several matching methods. Their approaches include (i) a greedy method that assigns the passenger request to its geometrically nearest taxi, (ii) a refined Hungarian method that finds a minimum cost bipartite matching between passenger request and taxis, and (iii) a bipartite matching method that minimizes the maximal cost of a matched request-taxi pair. Meanwhile, Tong et al. [4] also focused on a online minimum bipartite matching approach for taxi dispatches. They found that the greedy method in [3], which has an exponential competitive ratio, has a much better average performance than other bounded algorithms. Tian et al. [5] designed a real-time taxi dispatch system called Noah, depending on a shortest path algorithm, a dynamic matching algorithms, and a spatial taxis retrieving method. Zhang et al. [6] formulated a taxi dispatch optimization problem using integer linear programming, and derived the optimal solution under a small system scale. When the number of passenger requests and taxis becomes large, a heuristic algorithm was proposed to achieve a faster execution time. Ma et al. [7] studied taxi dispatch schedules, aim to minimize total travel distance via a spatio-temporal index. Traditional approaches may not be applied to the O2O model, since the interest of the driver may not align with the interest of the company.

In order to improve the taxi efficiency, carpool approaches become popular. Li et al. [8] considered a two-stage schedule in which passengers and parcels are cooperatively handled by the same taxi network. Its first stage routes passengers based on the Travelling Salesman Problem (TSP). Its second stage inserts parcels into the passenger route with minimum extra travel distances. Trasarti et al. [9] extracted mobility profiles of individuals from raw digital traces, and explored criteria to match individuals based on profiles. A matching criterion was developed to satisfy various basic constraints obtained from the background knowledge of the application domain. Santi et al. [10] proposed a method for shareability networks, which translate spatio-temporal sharing problems into a graph-theoretic framework with efficient solutions. Their experiments revealed the vast potential of routinely shareable taxis while keeping passenger discomfort low in terms of prolonged travel time. Zhang et al. [11] systematically designed a carpool service called coRide, which incorporates highly spatiotemporally correlated demand/supply models and real-time GPS location and occupancy information. It matches demand and supply for service quality with minimum taxi idle driving distance.

This paper is related to the Stable Marriage Problem (SMP). SMP aims to find a stable matching between two equally sized sets of elements (i.e., men and women), given complete preference orders of each man and woman [12]. Stability requires that a matched man and a matched woman will not prefer each other over their existing partners. This paper extends the SMP to the scenario with unequal numbers of passenger requests and taxis. The SMP has several important variations. Sethuraman et al. [13] incorporated geometric properties into the SMP, lead to the existence of median stable solutions, in which an element is assigned to their median stable partners. Iwama et al. [14] relaxed the SMP with incomplete preference orders and preference ties. If incomplete preference orders and preference ties are both allowed, the SMP becomes NP-hard, and no algorithm can guarantee a polynomial approximation ratio. Király [15] proposed a linear time local approximation algorithm for maximum stable marriage, which aims to find maximum number of stable matched pairs.

III. Model and Formulation

This section describes the taxi dispatch scenario, analyzes the O2O business model, and formulates the problem.

A. Scenario and Notations

Our scenario is a three-dimensional Euclidean surface that represents the city. Taxi dispatches are batched according to a discrete approach: time is discretized into frames (e.g., one minute per frame). Idle taxis are dispatched to passenger requests within the current frame. Let t_i denote the i-th idle taxi and its location in the current frame. $T = \{t_i\}$ is the set of taxis and their current locations. Let $r_j = (r_j^s, r_j^d)$ denote the j-th passenger request, with its pick-up and drop-off locations to be r_j^s and r_j^d, respectively. $R = \{r_j\}$ is the set of passenger requests. A distance function, $D(\cdot, \cdot)$, is introduced to measure the shortest path distance between different locations. For example, $D(t_i, r_j^s)$ represents the shortest path distance for the i-th taxi to pick up the j-th passenger request. $D(r_j^s, r_j^d)$ is the taxi traveling distance from picking up to dropping off the j-th passenger. The objective of this paper is to find a taxi dispatch schedule S, which maximally and stably matches T to R with balanced interests of passengers, drivers, and the company. Finally, let $S(r_j)$ be r_j's partner in S (i.e., the taxi that is dispatched to r_j in S).

B. Problem Formulation

In the O2O taxi business, passengers, drivers, and the company are three parties with different interests. The company (e.g., Uber Technologies) is assumed to have neutral interests with respect to passengers and taxi drivers. This is because the company is essentially a broker, who acts on behalf of both the passenger and the taxi driver in a ride. Consequently, the company makes money through taking a fixed percentage of the fare of each taxi ride [16]. The company aims to make the most money from fares of taxi rides.

This paper will discuss both non-sharing and sharing taxi dispatches. Depending on taxi sharing or not, the interests of passengers and taxi drivers are described later, respectively. As a result, passengers and taxi drivers will have preference orders for each other. A stable marriage approach is proposed to balance the interests of passengers and taxi drivers. The objective is to find a stable matching (taxi dispatch schedule) between passenger requests and taxis. Being different from the stable marriage problem, the numbers of passenger requests and taxis may not be the same (i.e., $|R| = |T|$ is not required). A dummy entry in the preference order represents no taxi dispatch. The number of stably matched passenger requests and taxis is maximized to fit the interest of the company.

IV. Non-Sharing Taxi Dispatching

This section focuses on non-sharing taxi dispatches. Each taxi can hold at most one passenger request, while a passenger request can ask for only one taxi.

A. Interests of Passengers and Taxi Drivers

Let us start with passengers. Usually, passengers will not care about the taxi brand or the gender of the driver. Instead, a passenger mainly cares about on the taxi wait time [17, 18]. As a result, the passenger's preference order depends on the distance between the taxi and the passenger (i.e., the nearest taxi is the best for the passenger). In other words, the passenger request r_j prefers the taxi $t_{i'}$ over t_i if $D(t_{i'}, r_j^s) < D(t_i, r_j^s)$. If t_i does not have enough seats for r_j, then r_j will put t_i to the end of its preference order. In this case, r_j can be divided into multiple requests (at the same location), each of which asks for a taxi with fewer seats.

In contrast to passengers, drivers have a more complicated interest model. Two parts should be considered: (i) the idle taxi driving distance from the current location of the taxi to the pick-up location of the passenger, and (ii) the taxi traveling distance from the passenger pick-up location to the passenger drop-off location. The first and second parts represent the expense and the pay-off of the taxi, respectively. We use a coefficient, α, to combine the above two parts. As a result, the taxi driver's preference order depends on $D(t_i, r_j^s) - \alpha D(r_j^s, r_j^d)$. The taxi driver t_i prefers the passenger request $r_{j'}$ over r_j, if $D(t_i, r_{j'}^s) - \alpha D(r_{j'}^s, r_{j'}^d) < D(t_i, r_j^s) - \alpha D(r_j^s, r_j^d)$. Note that the interests of passengers and taxi drivers may conflict with each other (t_i prefers r_j does not mean that r_j prefers t_i, and vice versa). An example is that the nearest taxi of a passenger may not prefer to pick up this passenger due to a low pay-off.

B. Matching Passenger Requests and Taxis

To match passenger requests and taxis, we need to extend the definition of matching stability:

Definition 1: A matching (i.e., taxi dispatch schedule) is stable, if an arbitrary matched passenger and another arbitrary matched driver will not prefer each other over their existing partners (including dummies partners, and dummies always prefer non-dummies over dummies).

Suppose that t_i is dispatched to r_j and $t_{i'}$ is dispatched to $r_{j'}$. A matching is not stable, if t_i prefers $r_{j'}$ over r_j and $r_{j'}$ prefers t_i over $t_{i'}$. Our stability definition is different from the traditional one, since it includes dummy partners (i.e., no dispatch scenario). This is because a passenger may not want a taxi if the taxi wait time is too long or the taxi does not have enough seats. Similarly, a taxi does not want to accept a passenger request that is too far away to make money. Note that dummies always prefer non-dummies over dummies. To obtain a stable matching, a taxi should always be matched to a passenger request that is more preferred over a dummy partner. Otherwise, the stability definition is not satisfied. In addition, this is similar for passenger requests.

A passenger request's preference order includes all taxis and an extra dummy entry (i.e., no dispatch). A taxi's preference order includes all passenger requests and also an extra dummy entry (i.e., no service). The numbers of passenger requests and taxis may not be equal, and the position of the dummy entry can be arbitrary in the preference order. We have:

Theorem 1: A stable matching exists, if exact one dummy entry (i.e., no dispatch) exists in the preference order of each passenger request and that of each taxi. This theorem validates even if $|R| \neq |T|$.

Proof: We prove by using dummy passenger requests and taxis. Let $|R|$ and $|T|$ denote the number of passenger requests and the number of taxis, respectively. $|T|$ dummy passenger requests and $|R|$ dummy taxis are added. $|T|$ dummy passenger requests will replace the dummy entry in the preference order of each taxi (each dummy entry becomes $|T|$ consecutive dummy passenger requests). Similarly, $|R|$ dummy taxis will replace the dummy entry in the preference order of each passenger request. The preference orders of dummy passenger requests and dummy taxis always prefer non-dummies over dummies. After adding dummy entries, our problem becomes a classic stable marriage problem with $|R| + |T|$ elements on each side. Consequently, a stable matching exists [12]. If we map dummy passenger requests and taxis back to the dummy entry, the matching stability maintains based on Definition 1. Therefore, the proof completes. ■

Algorithm 1 is proposed to find out the stable matching in Theorem 1. The key idea is to handle dummy entries as the end of the matching. If we go back to the proof of Theorem 1, once a passenger request or a taxi is matched with a dummy partner, it will not be matched to a non-dummy partner (otherwise it is not stable). Algorithm 1 consists of three parts. Lines 1 to 7 include the first part, which is a sub-algorithm called Proposal. Given a passenger request r_j, if the next entry in its preference order is non-dummy, r_j will propose a match to it

Algorithm 1 Non-Sharing Taxi Dispatch

Input: a set of taxis, T, and a set of passenger requests, R.
Output: a passenger-optimal taxi dispatch schedule, S^*.

```
1:  Sub-algorithm Proposal
2:  Input: passenger request $r_j$.
3:  if the next entry in $r_j$'s preference order is non-dummy (let $t_i$ denote this entry) then
4:      Update $r_j$'s current entry to be $t_i$.
5:      Call sub-algorithm Refusal for $S^*$, $t_i$, and $r_j$.
6:  else
7:      $r_j$ is unserved (no taxi dispatch), and update $S^*$.
8:  Sub-algorithm Refusal
9:  Input: schedule $S^*$, taxi $t_i$, and passenger request $r_j$.
10: if $t_i$ is undispatched and prefers $r_j$ over no dispatch then
11:     Dispatch $t_i$ to $r_j$, and update $S^*$.
12: else if $t_i$ is dispatched to $r_{j'}$, but prefers $r_j$ over $r_{j'}$ then
13:     Dispatch $t_i$ to $r_j$, and update $S^*$.
14:     Call sub-algorithm Proposal for $r_{j'}$.
15: else
16:     Call sub-algorithm Proposal for $r_j$.
17: Non-Sharing Taxi Dispatch
18: Compute preference orders for $\forall r_i \in R$ and $\forall t_j \in T$ based on $D(t_i, r_j^s)$ and $D(t_i, r_j^s) - \alpha D(r_j^s, r_j^d)$, respectively.
19: Initialize each taxi in $T$ as undispatched.
20: for each passenger request, $r_j \in R$ do
21:     Call sub-algorithm Proposal for $r_j$.
22: return $S^*$ as a passenger-optimal taxi dispatch schedule.
```

(lines 3 to 5). Otherwise, the algorithm terminates and no taxi is dispatched to r_j (lines 6 and 7). Lines 8 to 16 include the second part, which is a sub-algorithm called Refusal. When r_j proposes a match to t_i, t_i accepts the proposal if it is currently undispatched (lines 10 and 11). Otherwise, t_i will compare r_j with its current partner, keep the preferred one, and refuse the unpreferred one. The refused passenger request will go back to proposal (lines 12 to 16). Finally, lines 17 to 22 include the third part, which is the main program. Initializations are in lines 18 and 19. Dummy preference order entries are used if $D(t_i, r_j^s)$ and $\alpha D(t_i, r_j^s) - D(r_j^s, r_j^d)$ are larger than thresholds. If a taxi cannot provide enough seats for a passenger request, they are put to the end of the preference order of each other. In lines 20 and 21, each passenger request simply proposes to taxis one by one. The time complexity of Algorithm 1 is $O(|R| \cdot |T|)$. This is because each of $|R|$ passenger requests takes at most $|T|$ to exhaust its preference order.

To illustrate Algorithm 1, an example is shown in Fig. 2, in which grey grids are preference order entries that have been visited. Dummy entries are represented by $\emptyset$. Fig. 2(a) shows the preference orders of all passenger requests and taxis (left-side entries are prioritized over right-side entries). Taxis are initialized as undispatched. After the initialization, each passenger request proposes to taxis in turn. Fig. 2(b) shows r_1's proposal. The first entry of r_1 is t_1, and t_1 accepts r_1's

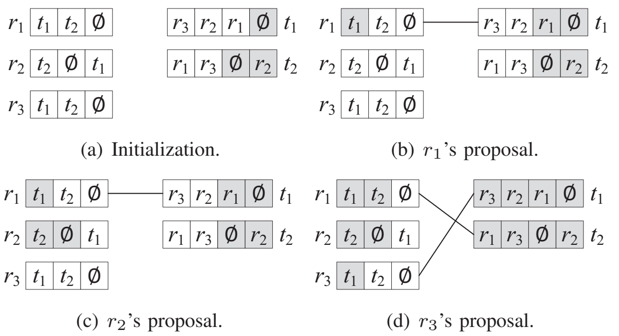

(a) Initialization. (b) r_1's proposal. (c) r_2's proposal. (d) r_3's proposal.

Fig. 2. An example to illustrate Algorithm 1.

proposal, since t_1 prefers r_1 over $\emptyset$. Fig. 2(c) shows r_2's proposal. The first entry of r_2 is t_2, however, t_2 prefers $\emptyset$ over r_2. As a result, r_2 is refused by t_2, and r_2 is undispatched since the next entry in r_2's preference order is $\emptyset$. Fig. 2(d) shows r_3's proposal. The first entry of r_3 is t_1, which has been matched to r_1. However, t_1 prefers r_3 over r_1. Therefore, t_1 accepts r_3's proposal, and refuses r_1. Consequently, r_1 re-proposes to the next entry in its preference order, t_2. Since t_2 prefers r_1 over $\emptyset$, it accepts r_1's proposal. Since r_1, r_2, and r_3 have completed their proposals, Algorithm 1 terminates (t_1 is dispatched to r_3, t_2 is dispatched to r_1, and r_2 is unserved).

C. Algorithmic Property

Algorithm 1 has several interesting properties:

Property 1: In the stable matching obtained by Algorithm 1, if a taxi prefers no dispatch over all passenger requests, then it will not be dispatched. Similarly, if a passenger request prefers no service over all taxis, then it will not be served.

When a taxi prefers no dispatch over all passenger requests, it will refuse all proposals from passenger requests. Similarly, when a passenger request prefers no service over all taxis, then it will not propose to any taxis. This property gives flexibilities to passengers and taxi drivers in the O2O business.

Property 2: Among all stable matchings, the stable matching obtained by Algorithm 1 satisfies that passenger requests have their best partners, but taxis have their worst partners.

This property is the same as the traditional Gale-Shapley algorithm [12] (i.e., passenger-optimal taxi dispatch schedule). As an extension, we have:

Theorem 2: In the passenger-optimal stable matching obtained by Algorithm 1, if a passenger request is unserved, then it is unserved in all possible stable matchings.

Proof: Let S^* denote the passenger-optimal stable matching obtained by Algorithm 1. r_j denotes a passenger request that is unserved in S^*. We prove by contradiction. Let S denote another stable matching, i.e., $S \neq S^*$. Let $S(r_j)$ denote the taxi that is matched to r_j in S. By the definition of stability, r_j must prefer $S(r_j)$ over a dummy in its preference order. However, $S(r_j)$ must have been proposed by r_j in Algorithm 1. This is because each passenger request proposes to more preferred taxis before less preferred taxis in Algorithm 1. Therefore, $S(r_j)$ must refuses r_j due to other proposals in S^*, leading to a contradiction. Therefore, Theorem 2 validates. ■

Algorithm 2 Non-Sharing Taxi Dispatch (All Schedules)

Input: a set of taxis, T, and a set of passenger requests, R.
Output: a set of all taxi dispatch schedules, S_a.

```
 1: Sub-algorithm BreakDispatch
 2: Input: a schedule, $S$, and a passenger request, $r_j$.
 3: Let $S(r_j)$ denote $r_j$'s partner in $S$. Break the matching between $S(r_j)$ and $r_j$. Under given rules, call sub-algorithm Proposal for $r_j$ to try to obtain a new schedule $S'$.
 4: if $S'$ is successfully obtained. then
 5:     Add $S'$ to the set of all taxi dispatch schedules, $S_a$.
 6:     for each passenger request, $r_{j'} \in |R|$ and $j' \geq j$ do
 7:         Recursively call BreakDispatch for $S'$ and $r_{j'}$.
 8: Non-Sharing Taxi Dispatch (All Schedules)
 9: Call Algorithm 1 to compute the passenger-optimal stable matching $S^*$. Initialize the set $S_a = \{S^*\}$.
10: for each passenger request, $r_j \in |R|$ do
11:     Call the sub-algorithm BreakDispatch for $S^*$ and $r_j$.
12: return $S_a$ as the set of all taxi dispatch schedules.
```

The intuition of Theorem 2 is that, in Algorithm 1, passenger requests can obtain their best partners among all possible stable matchings. Therefore, the passenger request, which is unserved in Algorithm 1, is not able to get a more preferred partner among other possible stable matchings.

D. All Stable Matchings

Property 2 shows that, among all possible stable matchings, Algorithm 1 is a passenger-optimal one: passenger requests have their best partners, but taxis have their worst partners. However, Algorithm 1 ignores the interest of the company. We propose that the company can pick a stable matching from all possible ones [19], such that the most money is made.

As a result, Algorithm 2 is proposed to find out all possible stable matchings. The key idea is to start with the passenger-optimal stable matching, and try to break this stable matching to obtain other stable matches. Algorithm 2 includes two parts. Lines 1 to 7 include the first part, which is a sub-algorithm called BreakDispatch. Given a taxi dispatch schedule S and a passenger request r_j, the sub-algorithm BreakDispatch tries to break the current matching of r_j in S. BreakDispatch involves existing sub-algorithms Proposal and Refusal in Algorithm 1. Some rules are used in breaking the existing stable matching, and are explained later (line 3). BreakDispatch may not be successful, depending on the rules. If BreakDispatch is successful, another stable matching can be obtained (line 5), and is recursively used to obtain subsequent new stable matchings (lines 6 and 7). Lines 8 to 12 include the second part, which is the main program. As an initialization, Algorithm 1 is called to obtain the passenger-optimal stable matching S^* (line 9). For S^* and each r_j, the sub-algorithm BreakDispatch is iteratively called to obtain other stable matchings (lines 10 and 11).

Since the key idea is to break an existing stable matching to obtain new ones, several rules are needed to guarantee the correctness and avoid the redundancy. Let S be a taxi dispatch schedule that is a stable matching. Let $S(r_j)$ be r_j's partner in S (i.e., the taxi that is dispatched to r_j in S). We have:

Rule 1: Given S and r_j, a successful BreakDispatch terminates, when $S(r_j)$ is dispatched to another passenger request $r_{j'}$ that is non-dummy, and $S(r_j)$ prefers $r_{j'}$ over r_j. Otherwise, BreakDispatch is unsuccessful.

This rule is used to guarantee the correctness:

Theorem 3: Given S and r_j, a successful BreakDispatch with Rule 1 results in another stable matching S' ($S' \neq S$).

Proof: All pairs of matched passenger request and taxi are stable if they are both unaffected by BreakDispatch. Before forcing r_j to take a less preferred taxi, r_j has already been refused by more preferred taxis, since these taxis have better partners than r_j. In BreakDispatch, taxis will continue to get more preferred passenger requests, otherwise the proposal is rejected. BreakDispatch terminates when either (i) some passenger requests are re-matched to dummy partners, or (ii) $S(r_j)$ is dispatched to $r_{j'}$ and $S(r_j)$ prefers $r_{j'}$ over r_j. The former case does not yield a stable matching at the end, since $S(r_j)$ is undispatched. In contrast, the latter case can yield a stable matching, and the proof completes. ■

Rule 2: Given S and r_j, a successful BreakDispatch (and its Proposal and Refusal) only involves the passenger request $r_{j'}$ with $j' \geq j$. BreakDispatch becomes unsuccessful, if it involves a passenger request $r_{j'}$ with $j' < j$.

This rule is used to void the redundancy:

Theorem 4: Any stable matching, other than the passenger-optimal one, could be obtained exactly once by recursively applying BreakDispatch with Rules 1 and 2 to the passenger-optimal stable matching.

Proof: The proof is divided into two parts. In the first part, we prove that any other stable matching can be obtained by recursively using BreakDispatch. In the second part, we prove that any other stable matching is obtained exactly once.

We start with the first part. Let $S(r_j)$ denote r_j's partner in S. Let S^* denote the passenger-optimal stable matching. S is another stable matching that is not passenger-optimal. Without loss of generality, $S^*(r_j) \neq S(r_j)$. By contradiction, we show that no passenger request $r_{j'}$ proposes to a less preferred partner than $S(r_{j'})$ during BreakDispatch on S^*. If this could happen, let $r_{j'}$ be the first one that is refused by $S(r_{j'})$ during BreakDispatch on S^*. Therefore, $S(r_{j'})$ must receive a proposal from a more preferred passenger request, say $r_{j''}$. Since $r_{j'}$ is the first passenger request that is refused by $S(r_{j'})$ during BreakDispatch on S^*, $r_{j''}$ prefers $S(r_{j'})$ over $S(r_{j''})$. Consequently, $r_{j''}$ and $S(r_{j'})$ prefer each other over their partners in S, leading to a contradiction that S is stable. Therefore, no passenger request $r_{j'}$ proposes to a less preferred partner than $S(r_{j'})$ during BreakDispatch on S^*. BreakDispatch on S^* leads to a stable matching S', in which no passenger request has a less preferred partner than in S. If $S' \neq S$, then we can recursively use BreakDispatch on an arbitrary passenger request, whose partner in S' is better than in S. The proof idea is the monotonicity: BreakDispatch on the passenger-optimal solution monotonically makes each passenger request to get a less preferred partner.

For the second part, we prove that any other stable matching is obtained exactly once. Given a stable matching S, we focus on two different sequences of BreakDispatch on S. Let us consider the first occasion, in which these two sequences of BreakDispatch are different. The first and second sequences operate on r_j and $r_{j'}$, respectively ($j' > j$). For the second sequence, Rule 2 prevents r_j from proposing again. Therefore, r_j has different partners in the first and second sequences. As a result, different stable matchings are obtained by these two sequences. Hence, any other stable matching is obtained exactly once, and the proof completes. ■

Rules 1 and 2 are applied to guarantee the correctness and avoid the redundancy, respectively. On the other hand, Theorem 2 shows that, in the passenger-optimal stable matching obtained by Algorithm 1, if a passenger request is unserved, then it is unserved in all stable matchings. To improve the algorithm efficiency, one more rule is added:

***Rule** 3:* Given S and r_j, BreakDispatch is unsuccessful if $S(r_j)$ is a dummy (i.e., r_j is unserved in S).

Rule 3 can be simply derived from Theorem 2. Note that Rule 3 is not necessary for Algorithm 2, but could improve the algorithm performance by cutting off useless branches. Rule 3 is actually a completion of Rule 1: if Rule 3 is not satisfied, no more stable matchings can be obtained.

To illustrate these rules, an example is shown in Fig. 3. As shown in Fig. 3(a), the passenger-optimal stable matching S^* is computed by Algorithm 1: t_1 is dispatched to r_1, t_2 is dispatched to r_2, and r_3 is unserved. As shown in Fig. 3(b), Algorithm 2 first tries to call BreakDispatch on the passenger-optimal stable matching S^* and r_1. r_1 is forced to take the next less preferred partner, which is t_2. Since t_2 prefers r_1 over its existing partner r_2, r_2 is refused by t_2 and proposes to t_1 instead. Note that r_2's proposal is allowed by Rule 2. Since t_1 prefers r_2 over r_1, t_1 accepts r_2's proposal. According to Rule 1, a stable matching is successfully obtained. BreakDispatch is called recursively, however, other stable matchings are not successfully obtained based on Rule 3. This is because next preference order entries of r_1 and r_2 are dummy. In the next step, Algorithm 2 tries to call BreakDispatch on S^* and r_2, as shown in Fig. 3(c). r_2 is forced to take the next less preferred partner, which is t_1. Since t_1 prefers r_2 over its existing partner r_1, r_1 is refused by t_1 and proposes to t_2 instead. However, only r_j with $j \geq 2$ can propose according to Rule 2. Therefore, BreakDispatch on S^* and r_2 is unsuccessful. Meanwhile, t_1 does not receive a proposal from a better partner based on Rule 1. Finally, Fig. 3(d) shows BreakDispatch on S^* and r_3. Since r_3 is not served, this BreakDispatch is also unsuccessful according to Rule 3. As a result, only one stable matching (t_1 is dispatched to r_2, t_2 is dispatched to r_1, and r_3 is unserved) is obtained other than S^*.

V. Sharing Taxi Dispatching

This section focuses on sharing taxi dispatches. Taxis can hold multiple passenger requests, but a passenger request can ask for only one one taxi. The interests of passengers and taxi drivers are consistently refined in this scenario.

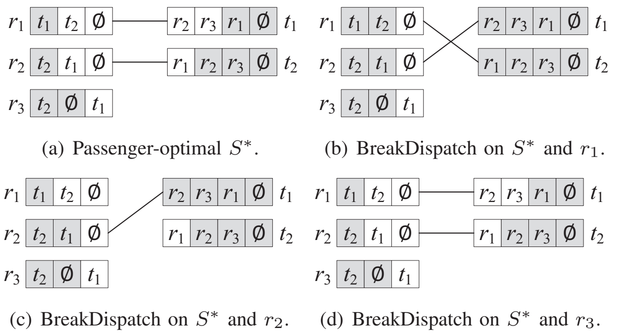

(a) Passenger-optimal S^*.
(b) BreakDispatch on S^* and r_1.
(c) BreakDispatch on S^* and r_2.
(d) BreakDispatch on S^* and r_3.

Fig. 3. An example to illustrate Algorithm 2.

A. Interests of Passengers and Taxi Drivers

Taxi sharing is a promising approach for saving commute cost while satisfying passengers' demands. It has been studied for several years to deal with people's routine commutes, and has been implemented as a part of the O2O taxi business such as UberPool. Recently it became more and more difficult for people to hail a taxi during rush hours in increasingly crowded urban areas. Naturally, taxi sharing is considered as a potential approach to tackle this emerging transportation issue. The key challenge is that the interests of passengers and drivers in the sharing taxi dispatch is different from those in the non-sharing taxi dispatch. Let $c_k = \{r_j, r_{j''}, ...\}$ denote a subset of passenger requests that share a taxi. We have:

***Theorem** 5:* Given c_k, it is NP-hard to compute a directed shortest path that goes through the pick-up location before the drop-off location for each passenger request in c_k.

Proof: We prove by reduction from the Shortest Hamiltonian Path Problem (SHPP) [20] in a weighted directed graph G. Given G, SHPP aims to find the shortest Hamiltonian path that visits each node exactly once. Note that SHPP keeps to be NP-complete when G is complete. This is because incomplete graphs are special cases of complete graphs with infinite edge weights for SHPP. By contradiction, we first claim that SHPP remains NP-complete in a complete graph G that has an even number of nodes. If SHPP is polynomially solvable in G with an even number of nodes, it is also solvable in G with an odd number of nodes through the following algorithm: (i) for each node in G, we can polynomially compute SHPP without this node, since the number of remaining nodes is even; (ii) for each node in G, we can exhaustively insert it to the SHPP of the remaining nodes, and thus obtain SHPP for G. Since G has an either odd or even number of nodes, SHPP in G becomes polynomially solvable, and thus, contradicts the truth. Hence, SHPP remains NP-complete in a complete graph G that has an even number of nodes. Given a complete graph G that has an even number of nodes, we can set the first half nodes in G's shortest Hamiltonian path to be pick-up locations of c_k, and set the remaining half nodes to be drop-off locations of c_k. At this time, the shortest Hamiltonian path reduces to a directed shortest path that goes through the pick-up location before the drop-off location for each passenger request in c_k. By reduction from the SHPP, the proof completes. ■

Theorem 5 shows the NP-hardness of designing a route for the taxi sharing among a subset of passenger requests. On the other hand, we observe that the number of passenger requests for a taxi sharing is usually no greater than three in our real life. Therefore, the practical route for the taxi sharing can be exhaustively searched. For example, given $c_k = \{r_j, r_{j'}, r_{j''}\}$, there exists in total $\frac{6!}{2!2!2!} = 90$ different feasible sequences of r^s_j, r^d_j, $r^s_{j'}$, $r^d_{j'}$, $r^s_{j''}$, and $r^d_{j''}$. An exhaustive search is acceptable to compute the directed shortest path that goes through the pick-up location before the drop-off location for c_k. Taxis are assumed to go through the above path for sharing among multiple passenger requests.

We revisit the interest of passengers, who mainly care about the taxi wait time. Let $D_{c_k}(t_i, r^s_j)$ denote the distance between t_i and r^s_j, when r_j shares t_i with passenger requests in c_k. Note that $D_{c_k}(t_i, r^s_j)$ is not the shortest path distance between t_i and r^s_j. Instead, it is the distance along c_k's shortest path (t_i could pick up other passenger requests before r_j). In addition, passengers may also care about the additional traveling distance due to taxi sharing. Similarly, let $D_{c_k}(r^s_j, r^d_j)$ denote the distance between r^s_j and r^d_j, when r_j shares t_i with other passenger requests in c_k. Therefore, $D_{c_k}(r^s_j, r^d_j) - D(r^s_j, r^d_j)$ is the additional traveling distance due to taxi sharing. As a result, the passenger request's preference order can depend on $D_{c_k}(t_i, r^s_j) + \beta[D_{c_k}(r^s_j, r^d_j) - D(r^s_j, r^d_j)]$, in which β is a coefficient to combine the taxi wait time and the additional traveling distance. A smaller value means that the taxi is more preferable. Note that when c_k includes only one passenger request r_j, $D_{c_k}(t_i, r^s_j) + \beta[D_{c_k}(r^s_j, r^d_j) - D(r^s_j, r^d_j)]$ reduces to $D(t_i, r^s_j)$, as used in no-sharing taxi dispatches. In addition, if several passenger requests share a taxi, their average preference value is used to rank taxis.

Taxi drivers also have a complicated interest model. Two parts are considered: (i) the total taxi driving distance including idle and non-idle driving distances, and (ii) the distance from each passenger's pick-up location to its drop-off location. The first and second parts represent the expense and the pay-off of the taxi, respectively. Let $D_{c_k}(t_i)$ denote the total taxi driving distance for t_i to serve all passenger requests in c_k. As a result, the taxi driver's preference order depends on $D_{c_k}(t_i) - (\alpha+1)\sum_{r_j \in c_k} D(r^s_j, r^d_j)$, in which α is a coefficient to combine the expense and the pay-off of the taxi. A notable point is that, when c_k includes only one passenger request r_j, $D_{c_k}(t_i) - (\alpha+1)\sum_{r_j \in c_k} D(r^s_j, r^d_j)$ can also reduce to $D(t_i, r^s_j) - \alpha D(r^s_j, r^d_j)$, as used in no-sharing taxi dispatches.

B. Matching Passenger Requests and Taxis

When taxis are shared, our taxi dispatch schedule includes two stages: (i) passenger requests are maximally packed to subsets according to their preferences, and (ii) each subset of passenger requests is regarded as a new passenger request to be independently scheduled by Algorithm 1. Since the second stage is trivial, we focus on the first stage. Let $C = \{c_k\}$ be the set of all feasible subsets of passenger requests that share a taxi. The feasibility means that, given a threshold θ, we have $D_{c_k}(r^s_j, r^d_j) - D(r^s_j, r^d_j) \leq \theta$ for $\forall r_j \in c_k, c_k \in C$. This is due to the interest of passengers, who share a taxi only if the additional traveling distance is limited. We assume that $2 \leq |c_k| \leq 3$ for practical usage, and thus, all feasible c_k can be computed through the exhaustive search. The first stage objective is to maximally pack passenger requests to feasible subsets. Let x_k be a boolean variable that indicates whether c_k is successfully packed. We have:

Algorithm 3 Sharing Taxi Dispatch

Input: a set of taxis, T, and a set of passenger requests, R.
Output: a taxi dispatch schedule, S.

```
1: Through exhaustive search on $R$, compute the set, $C$, of
   all feasible subsets of passenger requests that can share a
   taxi: $D_{c_k}(r^s_j, r^d_j) - D(r^s_j, r^d_j) \leq \theta$ for $\forall r_j \in c_k, c_k \in C$.
2: Given $C$, solve the MSPP in Eqs. 1 to 3 by the existing
   approximation algorithm in [21]. Let $C' = \{c_k \mid x_k = 1\}$
   and $R' = \{r_j \mid \sum_{k: r_j \in c_k} x_k = 0\}$.
3: Each subset of passenger requests in $C'$ is regarded as an
   independent request. Call Algorithm 1 on $T$ and $R' \cup C'$.
4: return the result for $T$ and $R' \cup C'$ (passenger requests
   in the same subset of $C'$ will share a taxi).
```

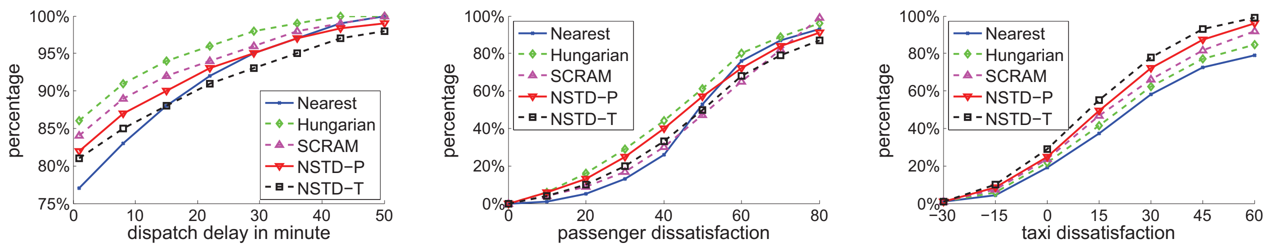
$$\text{maximize} \quad \sum_k x_k \tag{1}$$

$$\text{subject to} \quad \sum_{k: r_j \in c_k} x_k \leq 1 \text{ for } \forall j \tag{2}$$

$$x_k \in \{0, 1\} \text{ for } \forall k \tag{3}$$

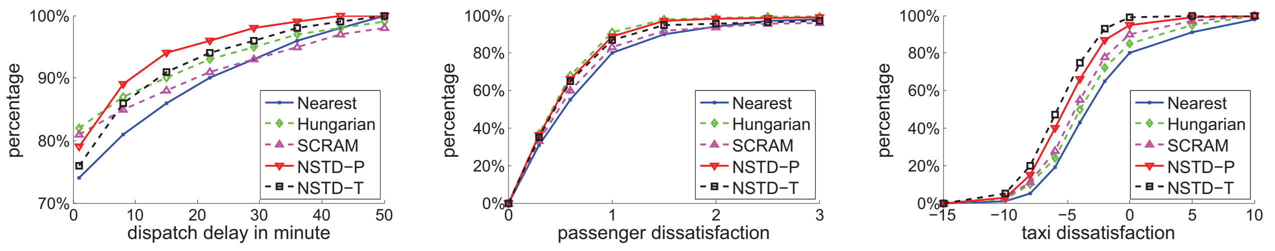
Eq. 1 is the objective of maximally packing passenger request to subsets. Eq. 2 is the constraint that each passenger request can be only packed to at most one subsets, since it will take at most one taxi. Eq. 3 is the boolean constraint for x_k. Eqs. 1 to 3 is essentially a Maximum Set Packing Problem (MSPP). There exists an approximation algorithm [21] that guarantees a ratio of $(\max_k |c_k| + 2)/3$ for MSPP. Since we assume $2 \leq |c_k| \leq 3$ for practical usage, this ratio is acceptable.

Hence, Algorithm 3 is proposed to compute the sharing taxi dispatch schedule. Line 1 conducts an exhaustive search on R to compute the set of all feasible subsets of passenger requests that can share a taxi. To meet the interests of passengers in taxi sharing, the constraint of $D_{c_k}(r^s_j, r^d_j) - D(r^s_j, r^d_j) \leq \theta$ for $\forall r_j \in c_k, c_k \in C$ is given. Due to the exhaustive search, line 1 takes a time complexity of $O(|R|^3)$. Given the set C, line 2 solves the MSPP through an existing approximation algorithm [21], taking $O(|C|^2) \subseteq O(|R|^2)$. Here, $C' = \{c_k \mid x_k = 1\}$ is a set of packed subsets, each of which can be regarded as an independent request to be scheduled by Algorithm 1. The passenger requests in the same subset of C' will share a taxi. Meanwhile, $R' = \{r_j \mid \sum_{k: r_j \in c_k} x_k = 0\}$ represents the set of passenger requests that are not included in any subsets of C'. Hence, each passenger request in R' will not share the taxi with others. Finally, line 3 calls Algorithm 1 on T and $R' \cup C'$ to dispatch taxis. A notable point is that the computations of preference orders become different, as previously described. The time complexity of Algorithm 3 is $O(|R|^3 + |R| \cdot |T|)$.



(a) Dispatch delay distribution. (b) Passenger dissatisfaction distribution. (c) Taxi dissatisfaction distribution.

Fig. 4. Algorithm performance for non-sharing taxi dispatches in the New York trace.



(a) Dispatch delay distribution. (b) Passenger dissatisfaction distribution. (c) Taxi dissatisfaction distribution.

Fig. 5. Algorithm performance for non-sharing taxi dispatches in the Boston trace.

VI. Experiments

Real data-driven experiments are conducted to evaluate the performances of the proposed algorithms. Results are shown from different perspectives to provide insights.

A. *Real Traces and Settings*

Experiments are conducted based on two real traces, the New York trace [22] and the Boston trace [23]. The New York trace was collected in January 2016, including 1,445,285 passenger requests. The Boston trace was collected in September 2012, including 406,247 passenger requests. Both traces include the request time, the pick-up and drop-off locations of each passenger request. According to the number of passenger requests, 700 and 200 taxis are simulated in the New York and Boston traces, respectively. Note that the New York trace includes the passenger requests in the New York state, and thus, this trace includes a significantly larger area than the Boston trace. The locations of taxis follow a two-dimensional normal distribution from the center of the city. Based on [24], the taxi speed is set to be 20 km/h within the trace. In addition, taxis are scheduled based on a one minute time frame.

B. *Comparison Algorithms and Metrics*

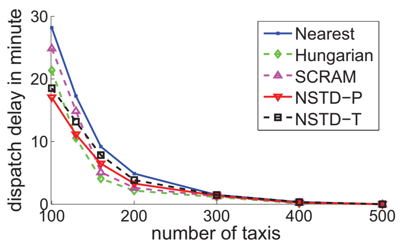
For non-sharing taxi dispatches, Algorithm 1 is denoted as NSTD-P, since it is passenger-optimal. Based on Algorithms 1 and 2, we compute the taxi-optimal stable matching, which is NSTD-T. Three comparison algorithms are used. (i) Nearest. It greedily dispatches the nearest idle taxi to a given passenger request. (ii) Hungarian. This is a modified Hungarian algorithm, in which the distances between passenger requests and taxis are matching costs. It returns a minimum cost matching. (iii) SCRAM [3]. It is also a matching algorithm, which minimizes the maximum cost for a matched pair.

For sharing taxi dispatches, Algorithm 3 is used with $\theta = 5$. We use STD-P and STD-T to denote the packed passenger-optimal and taxi-optimal stable matchings, respectively. Three comparison algorithms are used. (i) RAII [7]. It minimizes the total travel distance of taxis by using spatio-temporal indices to encode the location and time of passenger requests and taxis. (ii) SARP [8]. It is based on the TSP. It inserts new passenger requests to existing taxi routes to minimize the additional travel distance of taxis. (iii) Zhang [6]. It uses integer linear programming to solve the taxi sharing problem. A heuristic algorithm was proposed to achieve a faster execution time.

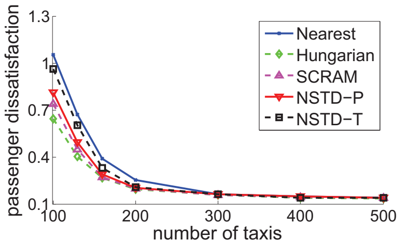
Three comparison metrics are employed. (i) Dispatch delay. It is the delay from the time that a passenger request is sent to the time that a taxi is dispatched to this passenger request. The taxi still needs some travel time before picking up the passenger. (ii) Passenger dissatisfaction: $D(t_i, r_j^s)$ for non-sharing taxi dispatches and $D_{c_k}(t_i, r_j^s)+\beta[D_{c_k}(r_j^s, r_j^d)-D(r_j^s, r_j^d)]$ for sharing taxi dispatches. β is set to be one. (iii) Taxi dissatisfaction: $D(t_i, r_j^s)-\alpha D(r_j^s, r_j^d)$ for non-sharing taxi dispatches and $D_{c_k}(t_i)-(\alpha+1)\sum_{r_j\in c_k} D(r_j^s, r_j^d)$ for sharing taxi dispatches. α is set to be one. For these metrics, smaller values represent better performances. The dissatisfaction unit is kilometers.

C. *Results for Non-Sharing Taxi Dispatches*

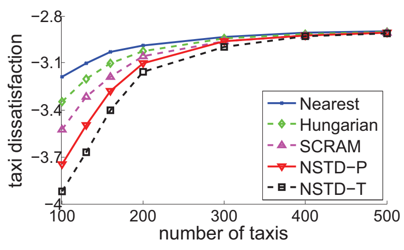
Results in the New York and Boston traces are shown in Figs. 4 and 5, respectively. Algorithm metrics are the dispatch delay, the passenger dissatisfaction, and the taxi dissatisfaction (smaller values represent better performances). We start with the New York trace in Fig. 4. Fig. 4(a) shows the Cumulative Distribution Function (CDF) of the dispatch delay. For all algorithms, more than 75% passenger requests receive taxi dispatch within one minute. Hungarian and SCRAM have the smallest dispatch delay, since they prioritize passengers. The proposed NSTD-P and NSTD-T have slightly larger dispatch



(a) Average dispatch delay.

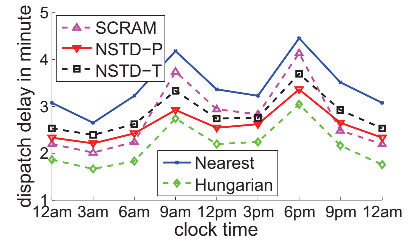


(b) Average passenger dissatisfaction.

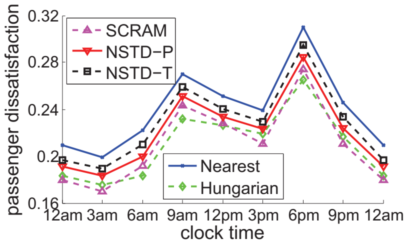


(c) Average taxi dissatisfaction.

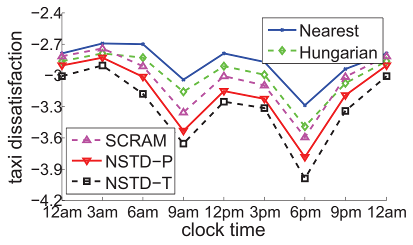
Fig. 6. Algorithm performance for non-sharing taxi dispatches in the Boston trace under different number of taxis.



(a) Average dispatch delay.



(b) Average passenger dissatisfaction.



(c) Average taxi dissatisfaction.

Fig. 7. Algorithm performance for non-sharing taxi dispatches in the Boston trace under different clock time.

delays. Nearest has the lowest percentage of dispatch delays within one minute, although almost all passenger requests have dispatch delays within 50 minutes. Fig. 4(b) shows the CDF of the passenger dissatisfaction. Hungarian and NSTD-P have the lowest passenger dissatisfaction. SCRAM and Nearest only have a small percentage of passengers with low dissatisfaction. However, the dissatisfaction of almost all passenger requests in SCRAM is lower than 80, since SCRAM minimizes the maximum cost among matched pairs. Fig. 4(c) shows the CDF of the taxi dissatisfaction. NSTD-P and NSTD-T significantly outperform the other algorithms, since they take the interests of taxi drivers into account. In contrast, the other algorithms only consider the interests of passengers.

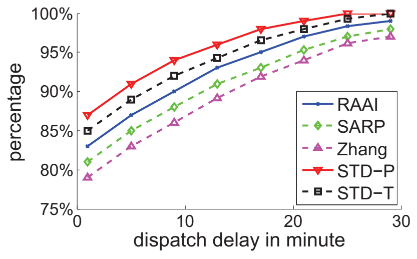
Fig. 5 shows similar results for the Boston trace: NSTD-P and NSTD-T have limited performance gaps to the literature in terms of the passenger dissatisfaction, but can significantly improve the taxi dissatisfaction. One difference in the Boston trace is that NSTD-P and NSTD-T are not outperformed in terms of the dispatch delay, as shown in Fig. 5(a). This is because they do not dispatch idle taxis to passengers that are too far away. In this case, passengers will wait for nearby busy taxis. Another difference is that the passenger and taxi dissatisfaction in the Boston trace is lower than that in the New York trace. This is because the New York trace involves a larger area than the Boston trace, leading to larger distances between passengers and taxis. Hence, the shapes of the CDFs in the Boston and New York traces are also different.

To study the impact of the number of taxis, Fig. 6 shows the relationship between the number of taxis and these three metrics (average values rather than CDFs) in the Boston trace. Fig. 6(a) shows that all the algorithms have larger dispatch delays when there are fewer taxis. This is because passengers have to wait idle taxis. The average dispatch delays of these algorithms are close to each other, even if the numbers of taxis vary. On the other hand, the passenger dissatisfaction becomes higher when there are fewer taxis, as shown in Fig. 6(b). This is because passengers are less likely to find nearby taxis when there are fewer taxis. Fig. 6(c) shows that NSTD-P and NSTD-T have great improvements in terms of the taxi dissatisfaction, especially when there are fewer taxis. The reason is that taxis in NSTD-P and NSTD-T can choose passengers, when there are fewer taxis (i.e., there are more passenger requests). We also study the impact of the clock time, as shown in Fig. 7. The pattern is significant for 9am and 6pm, when people travel between home and work place. In 9am and 6pm, there are more passenger requests (i.e., there are relatively fewer taxis), leading to a larger average dispatch delay, a higher average passenger dissatisfaction, and a lower taxi dissatisfaction.

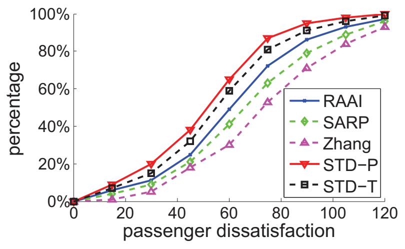
In summary, the proposed NSTD-P and NSTD-T have very limited performance gaps to the existing algorithms in terms of the dispatch delay and the passenger dissatisfaction. As a trade-off, they significantly outperform the existing algorithms in terms of the taxi dissatisfaction, especially when the number of taxis is fewer than the number of passenger requests or during the peak hours. Our approach considers the interests of both passengers and taxis, and refuses to dispatch taxis to passengers that are not preferred. In addition, NSTD-P and NSTD-T prioritize passengers and taxis, respectively.

D. Results for Sharing Taxi Dispatches

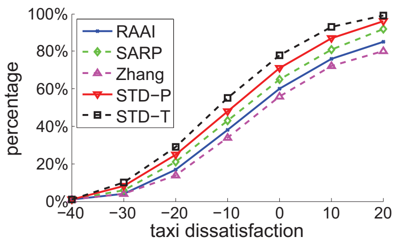
For sharing taxi dispatches, results in the New York and Boston traces are shown in Figs. 8 and 9, respectively. Being different from non-sharing taxi dispatches, STD-P and STD-T clearly outperform the comparison algorithms (RAAI, SARP, and Zhang), in terms of these three metrics. Although RAII minimizes the total travel distance of taxis, its spatio-temporal indices are information-lossy. SARP is based on the directed



(a) Dispatch delay distribution.

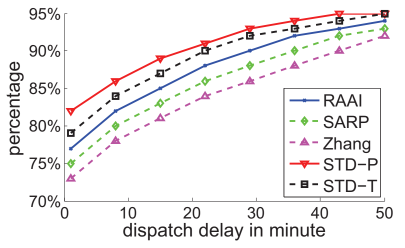


(b) Passenger dissatisfaction distribution.

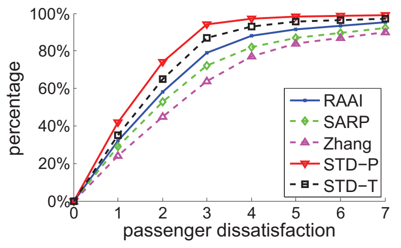


(c) Taxi dissatisfaction distribution.

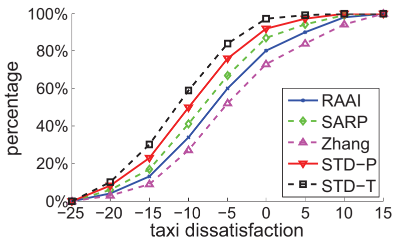
Fig. 8. Algorithm performance for sharing taxi dispatches in the New York trace.



(a) Dispatch delay distribution.



(b) Passenger dissatisfaction distribution.



(c) Taxi dissatisfaction distribution.

Fig. 9. Algorithm performance for sharing taxi dispatches in the Boston trace.

TSP. However, its approximation ratio is large, leading to a poor performance. Zhang uses the integer linear programming to pack passenger requests to a taxi, but it does not consider a good matching between them. Our approach firstly packs passenger requests that can share a taxi, and secondly uses stable matchings to balance the interests of passengers and taxi drivers, resulting in good performances. Another observation is that, compared to non-sharing taxi dispatches, sharing taxi dispatches have a smaller dispatch delay (since the number of passenger requests is fewer after packing), a higher passenger dissatisfaction (since passenger requests need to wait for the taxi to pick up), and a lower taxi dissatisfaction (since taxis have relatively longer distances to carry passengers).

VII. Conclusion

This paper focuses on the O2O taxi dispatches, in which the interests of passengers, drivers, and the company may not align with one another. To balance these interests, a stable marriage approach is proposed. The numbers of passenger requests and taxis can be different, and they can be matched to dummy partners. Both non-sharing and sharing taxi dispatches are studied. Experiments show the superior performances of our approach.

References

[1] N. Y. C. Taxi and L. Commission, *Taxi of tomorrow survey*, 2011.

[2] Y. A. Gonczarowski, N. Nisan, R. Ostrovsky, and W. Rosenbaum, “A stable marriage requires communication,” in *Proceedings of ACM-SIAM SODA*, 2015.

[3] J. P. Hanna, M. Albert, D. Chen, and P. Stone, “Minimum cost matching for autonomous carsharing,” *IFAC-PapersOnLine*, 2016.

[4] Y. Tong, J. She, B. Ding, L. Chen, T. Wo, and K. Xu, “Online minimum matching in real-time spatial data: experiments and analysis,” *Proceedings of the VLDB Endowment*, 2016.

[5] C. Tian, Y. Huang, Z. Liu, F. Bastani, and R. Jin, “Noah: a dynamic ridesharing system,” in *Proceedings of ACM SIGMOD*, 2013.

[6] S. Zhang, Q. Ma, Y. Zhang, K. Liu, T. Zhu, and Y. Liu, “QA-share: Towards efficient QoS-aware dispatching approach for urban taxi-sharing,” in *Proceedings of IEEE SECON*, 2015.

[7] S. Ma, Y. Zheng, and O. Wolfson, “T-share: A large-scale dynamic taxi ridesharing service,” in *Proceedings of IEEE ICDE*, 2013.

[8] B. Li, D. Krushinsky, H. A. Reijers, and T. Van Woensel, “The share-a-ride problem: People and parcels sharing taxis,” *European Journal of Operational Research*, 2014.

[9] R. Trasarti, F. Pinelli, M. Nanni, and F. Giannotti, “Mining mobility user profiles for car pooling,” in *Proceedings of ACM SIGKDD*, 2011.

[10] P. Santi, G. Resta, M. Szell, S. Sobolevsky, S. H. Strogatz, and C. Ratti, “Quantifying the benefits of vehicle pooling with shareability networks,” *Proceedings of the National Academy of Sciences*, 2014.

[11] D. Zhang, Y. Li, F. Zhang, M. Lu, Y. Liu, and T. He, “coRide: carpool service with a win-win fare model for large-scale taxicab networks,” in *Proceedings of ACM SenSys*, 2013.

[12] D. F. Manlove, R. W. Irving, K. Iwama, S. Miyazaki, and Y. Morita, “Hard variants of stable marriage,” *Theoretical Computer Science*, 2002.

[13] J. Sethuraman, C.-P. Teo, and L. Qian, “Many-to-one stable matching: geometry and fairness,” *Mathematics of Operations Research*, 2006.

[14] K. Iwama, S. Miyazaki, Y. Morita, and D. Manlove, “Stable marriage with incomplete lists and ties,” in *International Colloquium on Automata, Languages, and Programming*, 1999.

[15] Z. Király, “Linear time local approximation algorithm for maximum stable marriage,” *Algorithms*, 2013.

[16] J. V. Hall and A. B. Krueger, “An analysis of the labor market for ubers driver-partners in the united states,” *Princeton University Industrial Relations Section Working Paper*, 2015.

[17] L. Moreira-Matias, J. Gama, M. Ferreira, J. Mendes-Moreira, and L. Damas, “Predicting taxi–passenger demand using streaming data,” *IEEE Transactions on Intelligent Transportation Systems*, 2013.

[18] N. Wang and J. Wu, “Opportunistic WiFi offloading in a vehicular environment: waiting or downloading now?” in *Proceedings of IEEE INFOCOM*, 2016.

[19] D. McVitie and L. B. Wilson, “Stable marriage assignment for unequal sets,” *BIT Numerical Mathematics*, 1970.

[20] C. H. Papadimitriou, *Computational complexity*. John Wiley and Sons Ltd., 2003.

[21] M. Sviridenko and J. Ward, “Large neighborhood local search for the maximum set packing problem,” in *Proceedings of ICALP*, 2013.

[22] http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml.

[23] https://data.cityofboston.gov/Transportation/Boston-Taxi-Data/ypqb-henq/data.

[24] J. Chen, M. Weiszer, P. Stewart, and M. Shabani, “Toward a more realistic, cost-effective, and greener ground movement through active routingłpart i: Optimal speed profile generation,” *IEEE Transactions on Intelligent Transportation Systems*, 2016.