This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2760880, IEEE Transactions on Knowledge and Data Engineering

An Efficient Ride-Sharing Framework for Maximizing Shared Route

Na Ta Guoliang Li Tianyu Zhao Jianhua Feng Hanchao Ma Zhiguo Gong

Abstract—Ride-sharing (RS) has great values in saving energy and alleviating traffic pressure. Existing studies can be improved for better efficiency. Therefore, we propose a new ride-sharing model, where each driver has a requirement that if the driver shares a ride with a rider, the shared route percentage (i.e., the ratio of the shared route's distance to the driver's total travel distance) exceeds an expectation rate of the driver, e.g., 0.8. We consider two variants of this problem. The first considers multiple drivers and multiple riders and aims to compute driver-rider pairs to maximize the overall shared route percentage (SRP). We model this problem as the maximum weighted bigraph matching problem, where the vertices are drivers and riders, edges are driver-rider pairs, and edge weights are driver-rider's SRP. However it is rather expensive to compute the SRP values for large numbers of driver-rider pairs on road networks. To address this problem, we propose an efficient method to prune many unnecessary driver-rider pairs and avoid computing the SRP values for every pair. To improve the efficiency, we propose an approximate method with error bound guarantee. The basic idea is that we compute an upper bound and a lower bound for each driver-rider pair in constant time. Then we estimate an upper bound and a lower bound of the graph matching. Next we select some driver-rider pairs, compute their real shortest-route distance, and update the lower and upper bounds of the maximum graph matching. We repeat above steps until the ratio of the upper bound to the lower bound is not larger than a given approximate rate. The second considers multiple drivers and a single rider and aims to find the top-k drivers for the rider with the largest SRP. We first prune a large number of drivers that cannot meet the SRP requirements. Then we propose a best-first algorithm that progressively selects the drivers with high probability to be in the top-k results and prunes the drivers that cannot be in the top-k results. Extensive experiments on real-world datasets demonstrate the superiority of our method.

Index Terms—Ride-sharing, Shared Route Percentage, Bigraph Matching, Join-based Sharing, Search-based Sharing

1 Introduction

The sharing economy is booming with the Internet as the information carrier[6], [9]. In a sharing economy we aim to improve resource utilization by sharing the resource for multiple users. The shared resources include accommodation (e.g., Airbnb[1]), urban trip (e.g., Uber[3]), etc.

In this paper, we focus on the ride-sharing problem. We consider a new service called *hitch* in Uber[3] and Didi[2]. In the hitch service, there are a large number of part-time drivers, where each driver intends to drive from a source location to a destination and is willing to share the vehicle with a rider. If a driver shares the vehicle with a rider, the driver needs to first drive to the source of the rider from the driver's source, then drive to the destination of the rider from the rider's source (which is called the shared route between the driver and the rider), and finally drive to the destination of the driver from the rider's destination. The shared route percentage (SRP) is the ratio of the shared-route distance to the total route distance (i.e., the sum of the distances of the three parts). Since the driver must drive on the road networks, we consider the road-network distance and we assume that the driver will select the shortest route between two locations. Usually, the driver has a requirement that the SRP value must exceed an expectation value, e.g., 0.8, to guarantee a high sharing utility.

There are two variants of the ride-sharing (`RS`) problem. (1) `Join-basedRS`. There are a group of drivers and a group of riders and it aims to compute the driver-rider pairs that maximize the overall SRP. The `Join-basedRS` is suitable for peak-time traffic, where the numbers of riders and drivers are large and the join-based matching is a reasonable choice. (2) `Search-basedRS`. There are a group of drivers and a single rider, and it aims to compute the top-k drivers for the rider with the largest SRP values. The `Search-basedRS` can be used in the one-by-one serving mode where each rider expects instant response.

`Search-basedRS` optimizes the local matching for an individual rider while `Join-basedRS` optimizes the global matching for all riders. Our problems are different from traditional full-time driver services in Uber in that (i) those drivers are hired to complete riders' trips and drivers themselves have no traveling requirements; (ii) those servers aim to maximize the number of served riders while we aim to maximize the overall resource being shared by drivers and riders, because it is reasonable for drivers as resource (i.e., vehicles) owners to demand a higher resource sharing rate according to which they are compensated.

Example 1. *Figure 1 presents a running example. There are a group of drivers, d_1 - d_3 (source, destination, sharing requirement), and a group of riders, r_1 - r_4 (source, destination) on an urban road network. Solid lines are road segments and the numbers indicate road distances. For rider r_1, the SRP (0.524) for d_2 and r_1 does not meet d_2's requirement(0.6), therefore d_2 is an invalid driver for r_1. In contrast, the SRP requirements of both d_1 and d_3 are met, therefore they are valid drivers for r_1. We use the driver-rider pair $\langle d_1, r_1 \rangle$ to denote that driver d_1 picks up*

- *Na Ta is with School of Journalism and Communication, Renmin University. E-mail: tanayun@ruc.edu.cn.*
- *Guoliang Li, Tianyu Zhao and Jianhua Feng are with the Department of Computing Science, TNList, Tsinghua University, Beijing, China. {dan13, zhaoty17}@mails.tsinghua.edu.cn, {liguoliang, fengjh}@tsinghua.edu.cn. Guoliang Li is the corresponding author.*
- *Hanchao Ma is with the Department of Electronic Engineering and Computer Science, Washington State University. E-mail: mahanchao@hotmail.com.*
- *Zhiguo Gong is with the Department of Computer Science, Macau University. E-mail: fstzgg@umac.mo.*

1041-4347 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2760880, IEEE Transactions on Knowledge and Data Engineering

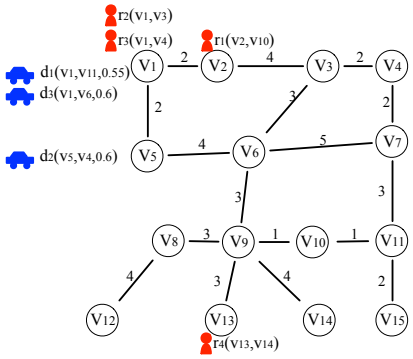


Fig. 1. Running Example.

rider r_1. For `Join-basedRS`*, we want to find a set of driver-rider pairs that maximize the overall SRP. $\langle d_1, r_1\rangle$, $\langle d_2, r_3\rangle$ and $\langle d_3, r_2\rangle$ are the optimal matching of drivers and riders that maximizes the overall SRP value. For* `Search-basedRS`*, given rider r_3, d_1, d_2 and d_3 are valid drivers, and d_2 is the top-1 driver with the largest SRP value.*

The performance is rather crucial in ride sharing because the riders will not wait for long time and we require to compute the driver-rider pairs within a minute[3], [2]. To meet such requirement, we propose an efficient ride-sharing framework. For `Join-basedRS`, we model this problem as the maximum weighted bigraph matching problem, where the vertices are drivers and riders, edges are driver-rider pairs, and edge weights are driver-rider' SRP values. However it is rather expensive to compute the SRP values for large numbers of driver-rider pairs on road networks. To address this problem, we propose an efficient method to prune many unnecessary driver-rider pairs and avoid computing the SRP values for such pairs. To further improve the efficiency, we propose an approximate method with any given approximate rate. The basic idea is that we compute an upper bound and a lower bound for each driver-rider pair in constant time. Then we estimate an upper bound and a lower bound of the maximum graph matching. Next we select some driver-rider pairs, compute their real shortest-route distances, and update the lower bound and upper bound of the graph matching. We repeat the above steps until the ratio of the upper bound to the lower bound is not larger than the given approximate rate. For `Search-basedRS`, we first prune a large number of drivers that cannot meet the SRP requirements. Then we propose a best-first method that progressively selects the drivers with high probability to be in the top-k results and prune the drivers that cannot be in the top-k results. Extensive experimental results on real-life datasets demonstrate the efficiency and quality of our method.

To summarize, we make the following contributions.

(1) We formally define the `Join-basedRS` and `Search-basedRS` problems and propose an effective and efficient ride-sharing framework.

(2) We devise an efficient exact algorithm to compute the optimal driver-rider pairs. We propose an approximate algorithm that can efficiently compute the driver-rider pairs with any given approximate rate.

(3) We develop a best-first method for the `Search-basedRS` problem to efficiently compute the top-k drivers for a rider.

(4) We have conducted an extensive set of experiments on real-world datasets. Experimental results show that our algorithms achieve high quality and efficiency.

The rest of the paper is organized as follows. The problem is formalized in Section 2. Section 3 introduces an exact algorithm for `Join-basedRS`. An approximate solution for `Join-basedRS` is developed in Section 4. Section 5 discusses the solution for `Search-basedRS`. Experimental results are reported in Section 6. Related works are reviewed in Section 7. Section 8 concludes the paper.

2 Problem Formulation

We first introduce the preliminaries (Section 2.1) and then formally define the problem of ride sharing (Section 2.2).

2.1 Preliminaries

We use a graph $G\langle V, E\rangle$ to model a road network, where each vertex $v \in V$ denotes a geo-location (e.g., road intersection), and each edge $(u, v) \in E$ is a road segment with an associated weight, e.g., road distance or traveling time. A route in a road network G is a connected path, and the route distance is the sum of the distance of each edge on the route. Given two vertices u, v, there are multiple routes between them and we assume that drivers will take the shortest route between the two vertices and use $\delta(u, v)$ to denote their shortest-route distance. We use route and distance to respectively refer to the shortest route and shortest-route distance for simplicity when the context is clear.

Definition 1 (Rider). *A rider $r_j = (r_j^{\triangleright}, r_j^{\triangleleft})$ plans to travel from source location $r_j^{\triangleright}$ to destination location $r_j^{\triangleleft}$ by taking a shared ride with a driver d_i.*

Definition 2 (Driver). *A driver $d_i = (d_i^{\triangleright}, d_i^{\triangleleft}, \pi_i)$ plans to travel from a source location $d_i^{\triangleright}$ to a destination location $d_i^{\triangleleft}$, and is willing to share the ride with a rider r_j if the shared route percentage $\pi(d_i, r_j)$ is above π_i, where,*

$$\pi(d_i, r_j) = \frac{\delta(r_j^{\triangleright}, r_j^{\triangleleft})}{\delta(d_i^{\triangleright}, r_j^{\triangleright}) + \delta(r_j^{\triangleright}, r_j^{\triangleleft}) + \delta(r_j^{\triangleleft}, d_i^{\triangleleft})} \quad (1)$$

Here we assume each driver shares the vehicle with only one rider, because (i) picking up multiple riders may violate the driver's own interest as the driver most likely works in a part-time basis at his/her own convenience to compensate the traffic cost, and (ii) it may not be user-friendly to share between multiple riders due to the detours incurred for picking up other riders. Suppose driver d_i shares the ride with rider r_j, they form a driver-rider pair $\langle d_i, r_j\rangle$. We want to quantify how d_i and r_j share the ride. The driver first drives to the rider's source location $r_j^{\triangleright}$ from his/her own source $d_i^{\triangleright}$, then drives to the rider's destination $r_j^{\triangleleft}$, and finally drives to his/her own destination $d_j^{\triangleleft}$. Thus there are three sub-routes traveled by d_i: the first is from $d_i^{\triangleright}$ to $r_j^{\triangleright}$, the second is the shared route of r_j and d_i, i.e., from $r_j^{\triangleright}$ to $r_j^{\triangleleft}$, and the last is from $r_j^{\triangleleft}$ to $d_i^{\triangleleft}$. The route from $r_j^{\triangleright}$ to $r_j^{\triangleleft}$ is shared by the driver and rider. Intuitively, the larger $\delta(r_j^{\triangleright}, r_i^{\triangleleft})$ the better. Thus we define the shared route percentage by Equation 1. A driver d_i will only share the ride with a rider when the shared percentage is above d_i's threshold π_i, where $\pi_i \in [0, 1]$.

We assume the source and destination locations of each driver and rider are on vertices in the road network. If the locations are not on vertices, we find their closest vertices. Since each road segment is very short, e.g., 50 meters, this method is widely used in existing works[38].

1041-4347 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2760880, IEEE Transactions on Knowledge and Data Engineering

TABLE 1
Notations.

Symbol	Meaning
$D = \{d_i\}$	The drivers set, $d_i = (d_i^{\triangleright}, d_i^{\triangleleft}, \pi_i)$
$R = \{r_j\}$	The riders set, $r_j = (r_j^{\triangleright}, r_j^{\triangleleft})$
$\langle d_i, r_j \rangle$	A driver-rider pair
$\pi(d_i, r_j)$	The shared route percentage of d_i and r_j
π_i	The SRP requirement of d_i
π_0	The minimum π_i, i.e., $\pi_0 = \min\{\pi_i\}$
$\delta(\cdot, \cdot)$	The shortest-route distance for two vertices
$\delta_j^{\pi_0}$	$(\frac{1}{\pi_0} - 1)\delta(r_j^{\triangleright}, r_j^{\triangleleft})$
P_x	A matching plan for `Join-basedRS`
P^*	The optimal plan for `Join-basedRS`
$\mathcal{C}$	$\{\langle d_i, r_j \rangle \mid \pi(d_i, r_j) \geq \pi_i, \forall d_i \in D, \forall r_j \in R\}$
$\mathcal{C}_j$	$\{d_i \mid \pi(d_i, r_j) \geq \pi_i, \forall d_i \in D\}$
$\mathcal{C}_j^k$	$\mathcal{C}_j^k \subseteq \mathcal{C}_j, \lvert\mathcal{C}_j^k\rvert = k$, $\forall d_{i^*} \in \mathcal{C}_j^k, \forall d_i \in \mathcal{C}_j \backslash \mathcal{C}_j^k, \pi(d_{i^*}, r_j) \geq \pi(d_i, r_j)$
$D_j^{\triangleright}$	$\{d_i \mid \delta(d_i^{\triangleright}, r_j^{\triangleright}) \leq \delta_j^{\pi_0}\}$ for r_j
$D_j^{\triangleleft}$	$\{d_i \mid \delta(d_i^{\triangleleft}, r_j^{\triangleleft}) \leq \delta_j^{\pi_0}\}$ for r_j
$\tilde{\mathcal{C}}_j$	$D_j^{\triangleright} \cap D_j^{\triangleleft}$
$\tilde{\mathcal{C}}$	$\cup_{r_j \in R}\{\langle d_i, r_j \rangle \mid d_i \in D_j^{\triangleright} \cap D_j^{\triangleleft}\}$

Example 2. *In Figure 1, the road network G has 15 vertices and 16 edges. There are three drivers: $d_1 = (v_1, v_{11}, 0.55)$, $d_2 = (v_5, v_4, 0.60)$ and $d_3 = (v_1, v_6, 0.60)$, and four riders: $r_1 = (v_2, v_{10})$, $r_2 = (v_1, v_3)$, $r_3 = (v_1, v_4)$ and $r_4 = (v_{13}, v_{14})$. The distance of r_1's route, $(v_2 v_3 v_6 v_9 v_{10})$, is 11. For driver-rider pair $\langle d_1, r_1 \rangle$, $\pi(d_1, r_1) = \delta(r_1^{\triangleright}, r_1^{\triangleleft})/(\delta(d_1^{\triangleright}, r_1^{\triangleright}) + \delta(r_1^{\triangleright}, r_1^{\triangleleft}) + \delta(r_1^{\triangleleft}, d_1^{\triangleleft})) = 11/(2 + 11 + 1) = 0.786$. Since $\pi(d_1, r_1) > \pi_1$, d_1 can share a ride with r_1.*

2.2 Problem Formulation

Next we formulate the ride sharing problem. Table 1 lists the notations used in this paper.

Join-based Ride Sharing. Firstly, we consider the case that there are many available drivers, and multiple riders want to take shared rides simultaneously. We aim to assign each rider r_j to a *valid driver* d_i where $\pi(d_i, r_j) \geq \pi_i$, in order to maximize the overall shared route percentage. Let $D = \{d_i\}$ and $R = \{r_j\}$ denote the set of drivers and the set of riders respectively. We use $\mathcal{C} = \{(d_i, r_j) \mid \pi(d_i, r_j) \geq \pi_i\}$ to denote the set of valid driver-rider pairs, and use $\mathcal{C}_j$ to denote the set of valid drivers for rider r_j.

Then we construct a weighted bigraph $\mathcal{G}(\mathcal{G}_R, \mathcal{G}_D, \mathcal{G}_E)$ based on the valid pairs, where $\mathcal{G}_R$ is the set of riders R and $\mathcal{G}_D$ is the set of drivers D. There is an edge between d_i and r_j if $\pi(d_i, r_j) \geq \pi_i$ and the weight is $\pi(d_i, r_j)$. Then we aim to find a subgraph from the bigraph, which corresponds to a set of driver-rider pairs such that one rider is assigned to at most one driver and one driver is allocated with at most one rider. We call such subgraph a matching plan.

Definition 3 (Matching Plan). *A matching plan P_x is a subgraph of $\mathcal{G}$ if no two edges share a common bigraph vertex.*

Obviously we aim to find the matching plan with the largest overall weight, i.e., the sum of weights of edges in the matching plan. We call such plan the *optimal* matching plan. Next we define the join-based ride-sharing problem.

Definition 4 (`Join-basedRS`). *Given a set of drivers $D = \{d_i\}$ and a set of riders $R = \{r_j\}$ on a road network G, find the optimal plan P^*, among all possible matching plans, that maximizes the following objective function:*

$$P^* = \arg\max_{P_x} \sum_{\forall \langle d_i, r_j \rangle \in P_x} \pi(d_i, r_j) \tag{2}$$

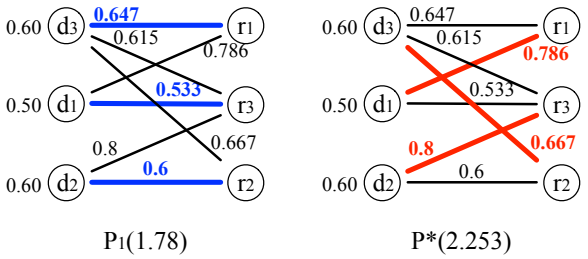


Fig. 2. Bigraph $\mathcal{G}$.

The complexity of join-based ride sharing is polynomial and we propose an algorithm in cubic time (Section 3). Since the algorithm cannot scale to large datasets, we propose an approximate solution in quasi-linear complexity (Section 4).

Example 3. *Given the drivers/riders sets in Figure 1, for `Join-basedRS`, Figure 2 shows a bigraph with valid pairs $\{\langle d_1, r_1 \rangle, \langle d_1, r_3 \rangle, \langle d_2, r_2 \rangle, \langle d_2, r_3 \rangle, \langle d_3, r_1 \rangle, \langle d_3, r_2 \rangle, \langle d_3, r_3 \rangle\}$. The numbers on edges are the shared route percentage values of the driver-rider pairs, and the number to the left of d_i is π_i. $P_1 = \{\langle d_1, r_3 \rangle, \langle d_2, r_2 \rangle, \langle d_3, r_1 \rangle\}$ is a matching plan, and the sum of the shared route percentages is $\pi(d_1, r_3) + \pi(d_2, r_2) + \pi(d_3, r_1) = 0.533 + 0.6 + 0647 = 1.78$. Rider r_4 has no valid driver. $P^* = \{\langle d_1, r_1 \rangle, \langle d_2, r_3 \rangle, \langle d_3, r_2 \rangle\}$ is the optimal matching plan with the largest weight, 2.253.*

Search-based Ride Sharing. Secondly, we consider the case that there are many drivers and a rider comes and requires to find top-k drivers with the largest SRP. The complexity of search-based ride sharing is polynomial and we propose an algorithm in quasi-linear complexity (Section 5).

Definition 5 (`Search-basedRS`). *For rider r_j on a road network G, find a k-size valid driver set $\mathcal{C}_j^k = \{d_{i^*}\}$,where*
(1) $\mathcal{C}_j^k \subseteq \mathcal{C}_j$, $\lvert\mathcal{C}_j^k\rvert = k$, and
(2) $\forall d_{i^*} \in \mathcal{C}_j^k, \forall d_i \in \mathcal{C}_j \backslash \mathcal{C}_j^k, \pi(d_{i^*}, r_j) \geq \pi(d_i, r_j)$.

Example 4. *Given the drivers/riders sets in Figure 1, for `Search-basedRS`, the valid driver set for rider r_3 is $\mathcal{C}_3 = \{d_1, d_2, d_3\}$, d_2 is the top-1 matching driver for r_3 since $\pi(d_2, r_3) > \pi(d_3, r_3) > \pi(d_1, r_3)$.*

3 Exact Method For Join-based RS

For `Join-basedRS`, a straightforward method enumerates every driver-rider pair, removes the invalid pairs, constructs a bigraph with the valid pairs, and utilizes the maximum weighted bigraph matching algorithm to compute the results. However it is rather expensive to compute the shortest route for every pair. Thus we propose an effective algorithm to prune large numbers of invalid pairs and avoid computing their shortest routes in Section 3.1. We then introduce a framework to utilize the bigraph matching algorithms to compute the optimal matching plan in Section 3.2.

3.1 Invalid Pair Pruning

For each rider r_j, a valid driver d_j should satisfy that $\pi(d_i, r_j) = \frac{\delta(r_j^{\triangleright}, r_j^{\triangleleft})}{\delta(d_i^{\triangleright}, r_j^{\triangleright}) + \delta(r_j^{\triangleright}, r_j^{\triangleleft}) + \delta(r_j^{\triangleleft}, d_i^{\triangleleft})} \geq \pi_i$. After rearranging the expression, we have

$$\delta(d_i^{\triangleright}, r_j^{\triangleright}) \leq (\frac{1}{\pi_i} - 1)\delta(r_j^{\triangleright}, r_j^{\triangleleft}) \tag{3}$$

$$\delta(r_j^{\triangleleft}, d_i^{\triangleleft}) \leq (\frac{1}{\pi_i} - 1)\delta(r_j^{\triangleright}, r_j^{\triangleleft}) \tag{4}$$

1041-4347 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2760880, IEEE Transactions on Knowledge and Data Engineering

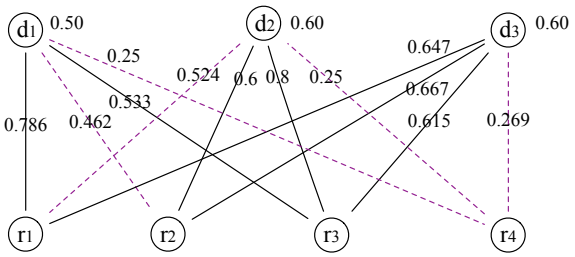


Fig. 3. The Driver-rider Pairs.

Therefore, we do not need to iterate over the whole driver set D, but only consider two much smaller subsets that satisfy equations 3 and 4 respectively, the intersection of which is the valid set of drivers for r_j. The network distance of the shared route between d_i and r_j, $\delta(r_j^{\triangleright}, r_j^{\triangleleft})$, can be calculated using any shortest route algorithm, e.g., Dijkstra or G-tree[38]. However, as π_i is driver-specific, the righthand sides of equations 3 and 4 are unknown given only rider r_j. To address this issue, we set a global threshold π_0, which is the minimum value among all drivers, i.e., $\pi_0 \leq \pi_i, \forall d_i \in D$. π_0 is practical as a driver is unlikely to share the trip if the route shared with a rider is only negligible. Then equations 3 and 4 are loosened into:

$$\delta(d_i^{\triangleright}, r_j^{\triangleright}) \leq (\frac{1}{\pi_0} - 1)\delta(r_j^{\triangleright}, r_j^{\triangleleft}) \tag{5}$$

$$\delta(r_j^{\triangleleft}, d_i^{\triangleleft}) \leq (\frac{1}{\pi_0} - 1)\delta(r_j^{\triangleright}, r_j^{\triangleleft}) \tag{6}$$

We use $D_j^{\triangleright}$ and $D_j^{\triangleleft}$ to denote driver sets that satisfy equations 5 and 6 for r_j respectively. Then we can construct a candidate set $\tilde{\mathcal{C}} = \{\langle d_i, r_j\rangle | d_i \in D_j^{\triangleright} \cap D_j^{\triangleleft}, \forall r_j \in R\}$. We can prove that $\tilde{\mathcal{C}}$ is *complete*, i.e., $\mathcal{C} \subseteq \tilde{\mathcal{C}}$. Theorem 1 guarantees the completeness of $\tilde{\mathcal{C}}$.

Theorem 1. *The candidate driver-rider set* $\tilde{\mathcal{C}} = \{\langle d_i, r_j\rangle | d_i \in D_j^{\triangleright} \cap D_j^{\triangleleft}, \forall r_j \in R\}$ *is complete, i.e.,* $\mathcal{C} \subseteq \tilde{\mathcal{C}}$, *where* $\mathcal{C} = \{\langle d_i, r_j\rangle | \pi(d_i, r_j) \geq \pi_i\}$.

Proof. We prove by contradiction. Suppose a rider r_j has a valid driver $d_{i'}$, it holds that $\langle d_{i'}, r_j\rangle \in \mathcal{C}$. If $\tilde{\mathcal{C}}$ did not contain $\langle d_{i'}, r_j\rangle$, it follows that $\delta(d_{i'}^{\triangleright}, r_j^{\triangleright}) > (\frac{1}{\pi_0} - 1)\delta(r_j^{\triangleright}, r_j^{\triangleleft})$, or $\delta(r_j^{\triangleleft}, d_{i'}^{\triangleleft}) > (\frac{1}{\pi_0} - 1)\delta(r_j^{\triangleright}, r_j^{\triangleleft})$.

If it is $\delta(d_{i'}^{\triangleright}, r_j^{\triangleright}) > (\frac{1}{\pi_0} - 1)\delta(r_j^{\triangleright}, r_j^{\triangleleft})$ (the hypothesis), given $\pi(d_{i'}, r_j) = \frac{\delta(r_j^{\triangleright}, r_j^{\triangleleft})}{\delta(d_{i'}^{\triangleright}, r_j^{\triangleright}) + \delta(r_j^{\triangleright}, r_j^{\triangleleft}) + \delta(r_j^{\triangleleft}, d_{i'}^{\triangleleft})} \geq \pi_{i'} \geq \pi_0$, it follows that $\frac{\delta(r_j^{\triangleright}, r_j^{\triangleleft})}{\delta(d_{i'}^{\triangleright}, r_j^{\triangleright}) + \delta(r_j^{\triangleright}, r_j^{\triangleleft})} \geq \pi_{i'} \geq \pi_0$, i.e., $\delta(d_{i'}^{\triangleright}, r_j^{\triangleright}) \leq (\frac{1}{\pi_0} - 1)\delta(r_j^{\triangleright}, r_j^{\triangleleft})$. This is contradictory with the hypothesis.

If it is $\delta(r_j^{\triangleleft}, d_{i'}^{\triangleleft}) > (\frac{1}{\pi_0} - 1)\delta(r_j^{\triangleright}, r_j^{\triangleleft})$ (the hypothesis), given $\pi(d_{i'}, r_j) \geq \pi_{i'} \geq \pi_0$, it follows that $\frac{\delta(r_j^{\triangleright}, r_j^{\triangleleft})}{\delta(r_j^{\triangleright}, r_j^{\triangleleft}) + \delta(r_j^{\triangleleft}, d_{i'}^{\triangleleft})} > \pi_i > \pi_0$. That is, $\delta(r_j^{\triangleleft}, d_{i'}^{\triangleleft}) \leq (\frac{1}{\pi_0} - 1)\delta(r_j^{\triangleright}, r_j^{\triangleleft})$. This is contradictory with the hypothesis.

Therefore, for any driver-rider pair $\langle d_{i'}, r_j\rangle \in \mathcal{C}$, it holds that $\langle d_{i'}, r_j\rangle \in \tilde{\mathcal{C}}$. Thus, $\mathcal{C} \subseteq \tilde{\mathcal{C}}$, and $\tilde{\mathcal{C}}$ is complete. □

Example 5. *In our running example, the straightforward method enumerates all the* 3×4 *driver-rider pairs, i.e.,* $\{\langle d_1, r_1\rangle, \langle d_1, r_2\rangle, \cdots, \langle d_3, r_3\rangle, \langle d_3, r_4\rangle\}$, *and computes all the pair-wise SRP to generate the candidate set as illustrated in Figure 3. Using equations 5 and 6, we get* $D_1^{\triangleright} = \{d_1, d_3\}$, $D_1^{\triangleleft} = \{d_1, d_3\}$ *for rider* r_1; $D_2^{\triangleright} = \{d_2, d_3\}$, $D_2^{\triangleleft} = \{d_2, d_3\}$ *for* r_2; $D_3^{\triangleright} = \{d_1, d_2, d_3\}$, $D_3^{\triangleleft} = \{d_1, d_2, d_3\}$ *for* r_3; *and* $D_4^{\triangleright} = \phi$, $D_4^{\triangleleft} = \phi$ *for* r_4. *So* $\tilde{\mathcal{C}} = (D_1^{\triangleright} \cap D_1^{\triangleleft}) \cup \cdots \cup (D_4^{\triangleright} \cap D_4^{\triangleleft}) = \{\langle d_1, r_1\rangle, \langle d_1, r_3\rangle, \langle d_2, r_2\rangle, \langle d_2, r_3\rangle, \langle d_3, r_1\rangle, \langle d_3, r_2\rangle, \langle d_3, r_3\rangle\}$. *We only need to compute the shared percentage for the seven pairs in* $\tilde{\mathcal{C}}$. *The computation cost is greatly reduced as a number of expensive shortest route calculations are avoided (the pairs connected with dotted lines in Figure 3).*

3.2 The Framework for `Join-basedRS`

We propose to compute the optimal matching plan.

(1) **Candidate Set Generation.**

(1.1) **Computing** $D_j^{\triangleright}$**.** For each rider r_j, we calculate $\delta_j^{\pi_0} = (\frac{1}{\pi_0} - 1)\delta(r_j^{\triangleright}, r_j^{\triangleleft})$ and compute the drivers whose distances from source locations to $r_j^{\triangleright}$ are within $\delta_j^{\pi_0}$, and the set of these drivers is exactly $D_j^{\triangleright}$, i.e., $D_j^{\triangleright} = \{d_i | \delta(d_i^{\triangleright}, r_j^{\triangleright}) \leq \delta_j^{\pi_0}\}$. To find such drivers, we can utilize the Dijkstra algorithm from $r_j^{\triangleright}$. We can also utilize an index based method. We first use the spatial index, e.g., grids or quadtree, to compute the drivers whose Euclidean distance from source locations to $r_j^{\triangleright}$ are within $\delta_j^{\pi_0}$. Then for each of such drivers, we utilize the road network index, e.g., G-tree, to compute its shortest route distance to $r_j^{\triangleright}$, i.e., $\delta(d_i^{\triangleright}, r_j^{\triangleright})$, and if the road distance is not larger than $\delta_j^{\pi_0}$, we add it into $D_j^{\triangleright}$.

(1.2) **Computing** $D_j^{\triangleleft}$**.** This process is similar to computing $D_j^{\triangleright}$. For each rider r_j, we calculate $\delta_j^{\pi_0}$ and compute the drivers whose distances to destination locations of $r_j^{\triangleleft}$ are within $\delta_j^{\pi_0}$, and the set of these drivers is exactly $D_j^{\triangleleft}$, i.e., $D_j^{\triangleleft} = \{d_i | \delta(r_j^{\triangleleft}, d_i^{\triangleleft}) \leq \delta_j^{\pi_0}\}$.

(1.3) **Computing** $\tilde{\mathcal{C}}$**.** Next, we compute the candidate driver set $\tilde{\mathcal{C}}_j$ of rider r_j by intersecting the two sets $D_j^{\triangleright}$ and $D_j^{\triangleleft}$, i.e. $\tilde{\mathcal{C}}_j = D_j^{\triangleright} \cap D_j^{\triangleleft}$. Then we can get the candidate driver-rider set $\tilde{\mathcal{C}} = \cup_{r_j \in R}\{\langle d_i, r_j\rangle | d_i \in \tilde{\mathcal{C}}_j\}$.

(2) **Constructing the bigraph.** For each pair $\langle d_i, r_j\rangle$ in $\tilde{\mathcal{C}}$, we compute the real SRP $\pi(d_i, r_j)$ of d_i and r_j. If $\pi(d_i, r_j) \geq \pi_i$, we add an edge between d_i and r_j in the bigraph.

(3) **Computing the optimal matching plan.** The `Join-basedRS` (see Definition 4) problem is essencially a maximum weighted bigraph matching problem [33]. We use the minimal cost network flow algorithm [4], [12] to compute the optimal matching plan P^*.

Algorithm 1 shows the pseudo code of the exact algorithm for `Join-basedRS`. The algorithm first uses function CANDIDATESETGEN (line 2) to generate the candidate set $\tilde{\mathcal{C}}$ of driver-rider pairs. For each rider r_j, function CANDIDATESETGEN computes $D_j^{\triangleright}$ and $D_j^{\triangleleft}$ using $\delta_j^{\pi_0}$ (lines 4-5), then intersects $D_j^{\triangleright}$ and $D_j^{\triangleleft}$ to identify candidate drivers for r_j and merges such candidate driver-rider pairs to produce $\tilde{\mathcal{C}}$ (line 6). Next, the algorithm builds the bigraph $\mathcal{G}$ from $\tilde{\mathcal{C}}$ (line 3), where an edge between driver d_i and rider r_j is added if $\pi(d_i, r_j) \geq \pi_i$. Finally, the algorithm employs graph matching techniques to compute the optimal matching plan (line 4).

Example 6. *Given the drivers/riders sets in Figure 1, the* CANDIDATESETGEN() *function first computes* $D_1^{\triangleright} = \{d_1, d_3\}$, $D_1^{\triangleleft} = \{d_1, d_3\}$, $\cdots$, $D_4^{\triangleright} = \phi$, $D_4^{\triangleleft} = \phi$ *for riders* r_1 *to* r_4. *Then* $\tilde{\mathcal{C}} = D_1^{\triangleright} \cap D_1^{\triangleleft} \cup \cdots \cup D_4^{\triangleright} \cap D_4^{\triangleleft} = \{\langle d_1, r_1\rangle, \langle d_1, r_3\rangle, \langle d_2, r_2\rangle, \langle d_2, r_3\rangle, \langle d_3, r_1\rangle, \langle d_3, r_2\rangle, \langle d_3, r_3\rangle\}$.

Then, the OPTIMALMATCH() *function computes the optimal matching plan for the bigraph* $\mathcal{G}$ *constructed from* $\tilde{\mathcal{C}}$. $P^* = \{\langle d_1, r_1\rangle, \langle d_2, r_3\rangle, \langle d_3, r_2\rangle\}$.

1041-4347 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2760880, IEEE Transactions on Knowledge and Data Engineering

Algorithm 1: ExactJoin$(R, D, G\langle V, E\rangle, \pi_0)$

Input: R: the riders set,
D: the drivers set,
$G\langle V, E\rangle$: the road network,
π_0: minimum SRP of all drivers
Output: P^*: the optimal matching plan

```
begin
    C̃ = CandidateSetGen(R, D, G⟨V, E⟩, π0);
    Build bigraph G from C̃;
    P* = OptimalMatch(G);
    return P*;
end
```

Function CandidateSetGen$(R, D, G\langle V, E\rangle, \pi_0)$

Input: R: the riders set, D: the drivers set,
$G\langle V, E\rangle$: the road network, π_0: minimum SRP
Output: $\tilde{\mathcal{C}}$: candidate set of driver-rider pairs

```
begin
    C̃ = {};
    for r_j ∈ R do
        Compute D_j^▷ using δ_j^{π0};
        Compute D_j^◁ using δ_j^{π0};
        C̃ = C̃ ∪ {⟨d_i, r_j⟩ | d_i ∈ D_j^▷ ∩ D_j^◁};
    return C̃;
end
```

Fig. 4. Exact Algorithm for Join-basedRS.

Driver Grouping. Given two drivers d_i and $d_{i'}$, if their source or destination locations are at the same road vertex, for each rider, we do not want to compute its shared route percentage with d_i and $d_{i'}$ repeatedly. To address this issue, we employ a list index structure to group drivers based on road vertices. For each road vertex v_a we create two lists: one is the source list $L^{\triangleright}(v_a)$, which contains drivers that starts from v_a, the other is the destination list $L^{\triangleleft}(v_a)$, which contains drivers that end their trips at v_a.

For rider r_j, if road vertex v_a is within $\delta_j^{\pi_0}$ distance to $r_j^{\triangleright}$, then all drivers in $L^{\triangleright}(v_a)$ are directly added to $D_j^{\triangleright}$; if road vertex v_a is within $\delta_j^{\pi_0}$ distance to $r_j^{\triangleleft}$, then all drivers in $L^{\triangleleft}(v_a)$ are directly added to $D_j^{\triangleleft}$. Therefore, we can avoid the duplicate computation of $\delta(d_i^{\triangleright}, r_j^{\triangleright})$ for every driver in $L^{\triangleright}(v_a)$, or $\delta(r_j^{\triangleleft}, d_i^{\triangleleft})$ for every driver in $L^{\triangleleft}(v_a)$.

Example 7. *Given the drivers/riders sets in Figure 1, for rider r_1, road vertex v_1 is within $\delta_1^{\pi_0}$ distance to $r_1^{\triangleright}$, therefore, the two drivers starting at v_1, d_1 and d_3, are added to $D_1^{\triangleright}$ together. Recall that in Example 6, we have to compute $\delta(d_1^{\triangleright}, r_j^{\triangleright})$ and $\delta(d_3^{\triangleright}, r_j^{\triangleright})$ and compare them to $\delta_1^{\pi_0}$ separately, which is unnecessary, and avoided by driver grouping.*

Complexity. We first analyze the complexity of computing $\tilde{\mathcal{C}}$. If we utilize the Dijkstra algorithm, the complexity is $\mathcal{O}(|E|\log(|V|))$. If we utilize the index, suppose the candidate set that uses the grid index to get the candidates within Euclidean distance $(\frac{1}{\pi_0} - 1)\delta(r_j^{\triangleright}, r_j^{\triangleleft})$ is $\tilde{\mathcal{C}}'$. The complexity to generate such candidates is $|\tilde{\mathcal{C}}'|$. Suppose we use the hierarchical structure G-tree to compute the shortest routes, where the road network is considered as a graph and partitioned into sub-graphs to store in tree nodes, with associating shortest inter-sub-graph distances. The assembly-based method is applied to ensure that the complexity is $\mathcal{O}(\log_f \frac{|V|}{l}|V|)$ (where f is the fan-out and l is the number of vertices in the leaf nodes of G-tree), and the complexity of computing $\tilde{\mathcal{C}}$ is $\mathcal{O}(\log_f \frac{|V|}{l}|V||\tilde{\mathcal{C}}|)$. The complexity of a bigraph matching algorithm is $\mathcal{O}(n^2 m)$, where n is the number of vertices, and m is the number of edges of the bigraph. Thus the complexity to find the optimal matching plan is $\mathcal{O}((|R| + |D|)^2|\tilde{\mathcal{C}}|)$. Accordingly, the overall complexity is $\mathcal{O}(|\tilde{\mathcal{C}}'| + \log_f \frac{|V|}{l}|V||\tilde{\mathcal{C}}| + (|R| + |D|)^2|\tilde{\mathcal{C}}|)$.

4 Approximate Method for Join-based RS

The exact method still needs to compute the SRP values for each valid driver-rider pair $\langle d_i, r_j\rangle$. It will be rather expensive if there are large numbers of valid pairs, because it requires to compute the shortest routes for these pairs. To address this issue, we propose an approximate method in Section 4.1 that uses upper/lower bounds of SRP (Section 4.2) to avoid the expensive shortest route computations and quickly generate the candidate set. It then gradually computes the real road-network distances for some pairs and shrinks the difference between the upper and lower bound until their ratio is not larger than a given threshold.

4.1 The Framework of the Approximate Method

Existing approximation method for maximum weight matching [11] are mainly designed for graphs with known edge weights, therefore, they are not suitable for providing approximate solusions for Join-basedRS, where we would like to compute as few edge weights as possible. Therefore, we propose a two-stage approximate method. First, we construct two bigraphs that bounds the accurate answer P^* within range $[\mathtt{LB}(P), \mathtt{UB}(P)]$. Then, we gradually update certain edges in the bound graphs into exact SRP values, so that $\mathtt{LB}(P)$ and $\mathtt{UB}(P)$ can converge to a predefined threshold τ, i.e., $\frac{\mathtt{UB}(P)}{\mathtt{LB}(P)} \leq \tau$. The final $\mathtt{LB}(P)$ value is the bounded approximate answer to P^*, and $\frac{P^*}{\mathtt{LB}(P)} \leq \tau$.

(1) **Constructing bound graphs.** For each driver-rider pair $\langle d_i, r_j\rangle \in \tilde{\mathcal{C}}$, we want to avoid the calculation of $\pi(d_i, r_j)$ that involves expensive shortest route computation. Instead, we devise a partition-based road network index to estimate the upper bound $\mathtt{UB}(\pi(d_i, r_j))$ and the lower bound $\mathtt{LB}(\pi(d_i, r_j))$ for each pair $\langle d_i, r_j\rangle$ (Section 4.2).

Then we construct two bigraphs using these bounds: the upper bound bigraph $\mathcal{G}^U(\mathcal{G}_R^U, \mathcal{G}_D^U, \mathcal{G}_E^U)$ and the lower bound bigraph $\mathcal{G}^L(\mathcal{G}_R^L, \mathcal{G}_D^L, \mathcal{G}_E^L)$, as follows:

(1.1) If $\mathtt{UB}(\pi(d_i, r_j)) < \pi_i$, d_i cannot be a valid driver for r_j, no edge is added to either $\mathcal{G}^U$ or $\mathcal{G}^L$.

(1.2) If $\mathtt{LB}(\pi(d_i, r_j)) \geq \pi_i$, d_i must be a valid driver for r_j, we add r_j to $\mathcal{G}_R^U$ and $\mathcal{G}_R^L$, add d_j to $\mathcal{G}_D^U$ and $\mathcal{G}_D^L$, and add an edge $\langle d_i, r_j\rangle$ into $\mathcal{G}_E^U$ and $\mathcal{G}_E^L$, where the weights of the edges are $\mathtt{UB}(\pi(d_i, r_j))$ and $\mathtt{LB}(\pi(d_i, r_j))$ respectively.

(1.3) Otherwise, $\mathtt{LB}(\pi(d_i, r_j)) < \pi_i \leq \mathtt{UB}(\pi(d_i, r_j))$, d_i can be a valid driver, it should be tested in order not to miss possible matching results, thus we apply the same logic to r_j, d_i and $\langle d_i, r_j\rangle$ as in case (1.2).

(2) **Incremental bounds convergence.**

(2.1) **Estimating initial upper bound for P^*.**

We first compute the sum of the maximum edge weight of each rider: $\mathtt{UB}(P_R) = \sum_{r_j \in \mathcal{G}_R^U} \max_{\langle d_i, r_j\rangle \in \mathcal{G}_E^U} \pi(d_i, r_j)$ using the upper bound graph; then we compute the

1041-4347 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2760880, IEEE Transactions on Knowledge and Data Engineering

sum of the maximum edge weight of each driver: $\texttt{UB}(P_D) = \sum_{d_i \in \mathcal{G}_D^U} \max_{\langle d_i, r_j \rangle \in \mathcal{G}_E^U} \pi(d_i, r_j)$ using the upper bound graph. Then the initial upper bound of P^* is: $\texttt{UB}(P) = \min\{\texttt{UB}(P_R), \texttt{UB}(P_D)\}$.

$\texttt{UB}(P)$ is a valid upper bound of P^* because when computing $\texttt{UB}(P_R)$ and $\texttt{UB}(P_D)$, the maximum edge weight of each rider and each driver is always used even if some maximum edges of different riders may have the same driver, or some maximum edges of different drivers may have the same rider, while in P^* such repetition is not allowed. Therefore, it holds that $\texttt{UB}(P_R) > P^*$, $\texttt{UB}(P_D) > P^*$, therefore, $\texttt{UB}(P) > P^*$.

(2.2) **Estimating initial lower bound for P^*.**

We perform a greedy algorithm over the riders set $\mathcal{G}_R^L$. For each rider r_j, we choose the edge between r_j and all the candidate drivers with the maximum weight, say $\langle d_m, r_j \rangle$, add the weight value $\texttt{LB}(\pi(d_m, r_j))$ to $\texttt{LB}(P)$. Then we remove r_j from $\mathcal{G}_R^L$ and d_m from $\mathcal{G}_D^L$, and their associated edges. This process is repeated until there is no edge in $\mathcal{G}_E^L$, and the acquired $\texttt{LB}(P)$ is the initial lower bound for P^*. $\texttt{LB}(P)$ is a valid lower bound of P^* because the best case result by the greedy algorithm is P^*, therefore, $\texttt{LB}(P) \leq P^*$.

(2.3) **Updating crucial edges.** For each edge $\langle d_i, r_j \rangle$, we compute the bounds difference: $\texttt{UB}(\pi(d_i, r_j)) - \texttt{LB}(\pi(d_i, r_j))$, and sort the differences of all edges in a descending order. Edges with higher bound differences are *crucial*: if their exact shared percentage values are computed and updated into $\mathcal{G}^U$ and $\mathcal{G}^L$, $\texttt{UB}(P)$ and $\texttt{LB}(P)$ can converge faster to each other. So we incrementally update h edges with the h largest bound differences each time. For each updated edge $\langle d_i, r_j \rangle$, if $\pi(d_i, r_j) < \pi_i$, the corresponding edges are removed from both $\mathcal{G}^U$ and $\mathcal{G}^L$. Otherwise, the corresponding edge weights in $\mathcal{G}^U$ and $\mathcal{G}^L$ are updated from $\texttt{UB}(\pi(d_i, r_j))$ and $\texttt{LB}(\pi(d_i, r_j))$ into $\pi(d_i, r_j)$.

(2.4) **Graph bounds convergence.** We re-calculate $\texttt{UB}(P)$ and $\texttt{LB}(P)$ using the updated edges. If $\frac{\texttt{UB}(P)}{\texttt{LB}(P)} \leq \tau$, the process terminates, and $\texttt{LB}(P)$ is the approximate answer for P^*. Otherwise, we repeat step (2.3), compute another h shared percentage values and update the bounds again, until $\frac{\texttt{UB}(P)}{\texttt{LB}(P)} \leq \tau$, or all edges in $\mathcal{G}^L$ are exact values (no approximate answer), for which the exact matching method can be applied to compute P^*.

Algorithm 2 shows the pseudo code of the approximate algorithm for `Join-basedRS`. The algorithm first creates a partition-based road network index (Section 4.2) to facilitate estimating the upper bounds and lower bounds of shared route percentage (line 2), then it builds the upper bound bigraph $\mathcal{G}^U$ and the lower bound bigraph $\mathcal{G}^L$ (line 3), where edges are the upper bounds and lower bounds of corresponding driver-rider pairs in $\tilde{\mathcal{C}}$. Next, the algorithm assigns the smaller value of $\texttt{UB}(P_R)$ and $\texttt{UB}(P_D)$ as the upper bound $\texttt{UB}(P)$ of matching result P^*, and the greedy matching result $\texttt{LB}(P)$ as the lower bound of P^* (line 4). During the bound convergence (lines 5-9), weights of h crucial edges are updated to exact SRP values in both $\mathcal{G}^U$ and $\mathcal{G}^L$ (line 6), $\texttt{UB}(P)$ and $\texttt{LB}(P)$ are updated accordingly (line 7). If $\texttt{UB}(P)$ and $\texttt{LB}(P)$ can converge to a ratio within τ, $\texttt{LB}(P)$ is returned as the approximate answer (line 9); otherwise, the exact matching method is used to compute P^* (line 10).

Algorithm 2: $\texttt{ApprJoin}(R, D, G, \pi_0, \tau)$

Input: R: the riders set, D: the drivers set,
G: the road network,
π_0: minimum SRP of all drivers,
τ: bigraph bound threshold
Output: $\texttt{LB}(P)$: the approximate answer of P^*

```
1  begin
2      Create the partition-based road network index;
3      Build upper/lower bound bigraphs: G^U, G^L;
4      Compute UB(P) and LB(P);
5      while UB(P)/LB(P) > τ and G^L has not-updated edges do
6          Update h crucial edges in G^U and G^L;
7          Compute UB(P) and LB(P);
8          if UB(P)/LB(P) ≤ τ then
9              return LB(P);
10     return ExactJoin(R, D, G⟨V, E⟩, π_0);
11 end
```

Fig. 5. Approximate Algorithm for `Join-basedRS`.

Example 8. *We demonstrate how Algorithm 2 works given the drivers/riders sets in Figure 1. We first construct the lower bound and upper bound bigraph using bounds of each driver-rider pair in $\tilde{\mathcal{C}}$. The edges in $\mathcal{G}^U$ and $\mathcal{G}^L$ are the same: $\{\langle d_1, r_1 \rangle, \langle d_1, r_3 \rangle, \langle d_2, r_2 \rangle, \langle d_2, r_3 \rangle, \langle d_3, r_1 \rangle, \langle d_3, r_2 \rangle, \langle d_3, r_3 \rangle\}$, with edge weights being $\texttt{UB}(\pi(d_i, r_j))$ and $\texttt{LB}(\pi(d_i, r_j))$ respectively. Initially $\texttt{UB}(P) = 2.4$, $\texttt{LB}(P) = 1.6$. Suppose $\tau = 1.3$, $h = 1$. Since $\frac{\texttt{UB}(P)}{\texttt{LB}(P)} = 1.5 > \tau$, we select h crucial edge $\langle d_2, r_2 \rangle$, the real $\pi(d_2, r_2) = 0.6$, and update corresponding edge weights in $\mathcal{G}^U$ and $\mathcal{G}^L$. Next we re-evaluate the graph bounds and get $\texttt{UB}(P) = 2.3$, $\texttt{LB}(P) = 1.8$. Now $\frac{\texttt{UB}(P)}{\texttt{LB}(P)} = 1.28 < \tau$, the process terminates and the approximate answer is 1.8. $\frac{P^*}{\texttt{LB}(P)} = \frac{2.253}{1.8} = 1.25 < \tau$.*

Complexity. It takes $\mathcal{O}(|\mathcal{G}_E^U|)$ and $\mathcal{O}(|\mathcal{G}_E^L|)$ time to build the upper and lower bound bigraphs. The time consumption of bound estimation is $\mathcal{O}(\max\{|R|, |D|\})$ for $\texttt{UB}(P)$, and $\mathcal{O}(|R|)$ for $\texttt{LB}(P)$. Suppose the algorithm terminates in y iterations and the complexity to compute the shortest route between a pair is $\mathcal{O}(\log_f \frac{|V|}{l}|V|)$ by the G-tree algorithm [38]. Then the update cost is $\mathcal{O}(yh \log_f \frac{|V|}{l}|V|)$. The total complexity is $\mathcal{O}(yh \log_f \frac{|V|}{l}|V| + y(|R| + |D|))$.

4.2 Bound Estimation

4.2.1 Upper/Lower Bounds of Network Distance

Lower Bound of Distance of $\delta(r_j^{\triangleright}, r_j^{\triangleleft})$. We can utilize the Euclidean distance to compute the lower bound of $\delta(r_j^{\triangleright}, r_j^{\triangleleft})$, i.e., $\texttt{LB}(\delta(r_j^{\triangleright}, r_j^{\triangleleft}))$, for any two locations. The time and space complexities are both $\mathcal{O}(1)$.

Upper Bound of Distance of $\delta(r_j^{\triangleright}, r_j^{\triangleleft})$. We first partition the road network G into a set of sub-graphs, and each sub-graph has a pivotal vertex. For each sub-graph, we maintain the minimum distance from each vertex to the pivotal vertex. For any two sub-graphs, we keep the distance between their pivotal vertices. Then given any two vertices, if they are in the same subgraph, we can utilize the sum of their distances to the pivotal vertex as the upper bound based on the triangle inequality. If they are in different subgraphs, we can utilize the sum of their distances to their pivotal

1041-4347 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2760880, IEEE Transactions on Knowledge and Data Engineering

vertices and the distance of the two pivotal vertices based on the triangle inequality. Next we formally introduce how to estimate an upper bound.

Graph partition. We use existing graph partition technique (e.g., Metis[21]) to partition the road network $G\langle V, E\rangle$ into a set of disjoint sub-graphs: $\mathcal{S}_G = \{G_p\langle V_p, E_p\rangle | 1 \leq p \leq |\mathcal{S}_G|\}$ with the following properties: (1) $V = \cup_{1\leq p\leq|\mathcal{S}_G|} V_p$, (2) for any two sub-graphs G_p and G_q, $V_p \cap V_q = \phi$, and (3)$E_p = \{(u, v)|u, v \in V_p, (u, v) \in E\}$.

Sub-graph distance array. We find a pivotal vertex c_p for each sub-graph G_p, whose maximum distance to any other vertices is minimum among all vertices in the sub-graph, i.e., $c_p = \arg\min_{v\in V_p} \max_{v'\in V_p, v\neq v'} \delta(v, v')$. Then we construct the $|V_p|$-sized distance array for G_p, each array element is the distance from a graph vertex v_s in V_p to c_p, i.e., $a_{v_s} = \delta(c_p, v_s)$ for $v_s \in V_p$.

Inter-sub-graph distance matrix. For all the sub-graphs, we build a distance matrix M, and each matrix element $m_{pq} = \delta(c_p, c_q)$ is the distance between the pivotal vertex c_p of sub-graph G_p and the pivotal vertex c_q of sub-graph G_q.

Distance upper bound. The upper bound of the network distance $\delta(v_s, v_t)$ between vertices v_s and v_t can be estimated as follows.

$$\texttt{UB}(\delta(v_s, v_t)) = \begin{cases} a_{v_s} + a_{v_t} & v_s, v_t \text{ in same sub-graph} \\ a_{v_s} + m_{pq} + a_{v_t} & v_s \in V_p, v_t \in V_q, p \neq q \end{cases} \tag{7}$$

Note the the upper bound can also be used to generate the candidate sets $D_j^{\triangleright}$ and $D_j^{\triangleleft}$ in equations 5 and 6. We can use $(\frac{1}{\pi_0} - 1)\texttt{UB}(\delta(r_j^{\triangleright}, r_j^{\triangleleft}))$ instead of $(\frac{1}{\pi_0} - 1)\delta(r_j^{\triangleright}, r_j^{\triangleleft})$. Although the candidate set may be enlarged as the right-hand sides of equations 5 and 6 are enlarged into $(\frac{1}{\pi_0} - 1)\texttt{UB}(\delta(r_j^{\triangleright}, r_j^{\triangleleft}))$, the verification time of extra candidate pairs can be efficiently reduced by avoiding the expensive shortest route computation, as verified by the experimental results in Section 6.

Example 9. *The road network in our running example can be partitioned into three sub-graphs G_1, G_2, and G_3, as illustrated in Figure 6. The pivotal vertices (bold), the distance arrays and matrix are sequentially established. For vertices $v_1, v_7 \in G_1$, $\delta(v_1, v_7) = 10$, $\texttt{UB}(\delta(v_1, v_7)) = a_{v_1} + a_{v_7} = 6 + 5 = 11$. For vertices $v_2 \in G_1$ and $v_{10} \in G_3$, $\delta(v_2, v_{10})) = 11$, $\texttt{UB}(\delta(v_2, v_{10})) = a_{v_2} + m_{13} + a_{v_{10}} = 7 + 5 + 1 = 13$.*

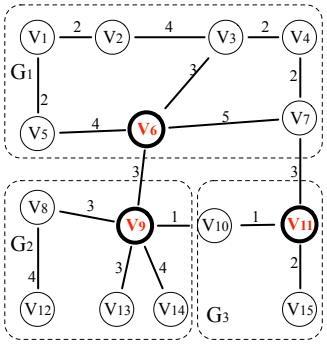
Complexity of the Index Structure. The distance array and pivotal vertex of each sub-graph G_p take $\mathcal{O}(|V_p|)$ space, and the distance matrix takes $\mathcal{O}(|\mathcal{S}_G|^2)$ where $|\mathcal{S}_G|$ is the number of sub-graphs. Therefore, the overall space complexity is $\sum_{G_p\in\mathcal{S}_G} \mathcal{O}(|V_p|) + \mathcal{O}(|\mathcal{S}_G|^2) = \mathcal{O}(|V|) + \mathcal{O}(|\mathcal{S}_G|^2)$. It takes $\mathcal{O}(|V_p|^3)$ time to construct the sub-graph distance array for a sub-graph V_p, and $\mathcal{O}(|\mathcal{S}_G|^2)$ time to construct the inter-sub-graph distance matrix. Since we focus on the static problem, there is no maintenance cost for different drivers/riders sets.

4.2.2 Upper/Lower Bounds of SRP of $\langle d_i, r_j\rangle$

We discuss how to compute the upper and lower bounds for SRP of a driver-rider pair $\langle d_i, r_j\rangle$.

Firstly, based on the definition of SRP, we have

$$\begin{aligned} \pi(d_i, r_j) &= \frac{\delta(r_j^{\triangleright}, r_j^{\triangleleft})}{\delta(d_i^{\triangleright}, r_j^{\triangleright}) + \delta(r_j^{\triangleright}, r_j^{\triangleleft}) + \delta(r_j^{\triangleleft}, d_i^{\triangleleft})} \\ &\leq \frac{\delta(r_j^{\triangleright}, r_j^{\triangleleft})}{\texttt{LB}(\delta(d_i^{\triangleright}, r_j^{\triangleright})) + \delta(r_j^{\triangleright}, r_j^{\triangleleft}) + \texttt{LB}(\delta(r_j^{\triangleleft}, d_i^{\triangleleft}))} \end{aligned} \tag{8}$$



c_1=v_6, A_1=[6, 7, 3, 5, 4, 0, 5]
c_2=v_9, A_2=[3, 0, 7, 4, 4]
c_3=v_{11}, A_3=[1, 0, 2]

M	v_6	v_9	v_{11}
v_6	0	3	5
v_9	3	0	2
v_{11}	5	2	0

Fig. 6. Index for Upper Bound.

For the righthand side of Equation 8, as $\texttt{LB}(\delta(d_i^{\triangleright}, r_j^{\triangleright})) + \texttt{LB}(\delta(r_j^{\triangleleft}, d_i^{\triangleleft})) \geq 0$, the numerator is no greater than the denominator. If we enlarge $\delta(r_j^{\triangleright}, r_j^{\triangleleft})$ in the numerator and the denominator into $\texttt{UB}(\delta(r_j^{\triangleright}, r_j^{\triangleleft}))$ simultaneously, the increment of the numerator is greater than the increment of the denominator, thus

$$\pi(d_i, r_j) \leq \frac{\texttt{UB}(\delta(r_j^{\triangleright}, r_j^{\triangleleft}))}{\texttt{LB}(\delta(d_i^{\triangleright}, r_j^{\triangleright})) + \texttt{UB}(\delta(r_j^{\triangleright}, r_j^{\triangleleft})) + \texttt{LB}(\delta(r_j^{\triangleleft}, d_i^{\triangleleft}))} \tag{9}$$

Therefore, we can establish an **upper bound** of $\pi(d_i, r_j)$:

$$\texttt{UB}(\pi(d_i, r_j)) = \frac{\texttt{UB}(\delta(r_j^{\triangleright}, r_j^{\triangleleft}))}{\texttt{LB}(\delta(d_i^{\triangleright}, r_j^{\triangleright})) + \texttt{UB}(\delta(r_j^{\triangleright}, r_j^{\triangleleft})) + \texttt{LB}(\delta(r_j^{\triangleleft}, d_i^{\triangleleft}))} \tag{10}$$

On the other hand, we have

$$\pi(d_i, r_j) \geq \frac{\delta(r_j^{\triangleright}, r_j^{\triangleleft})}{\texttt{UB}(\delta(d_i^{\triangleright}, r_j^{\triangleright})) + \delta(r_j^{\triangleright}, r_j^{\triangleleft}) + \texttt{UB}(\delta(r_j^{\triangleleft}, d_i^{\triangleleft}))} \tag{11}$$

If we shrink $\delta(r_j^{\triangleright}, r_j^{\triangleleft})$ in the numerator and the denominator of Equation 11 into $\texttt{LB}(\delta(r_j^{\triangleright}, r_j^{\triangleleft}))$ simultaneously, the reduction of the numerator is smaller than the reduction of the denominator, thus

$$\pi(d_i, r_j) \geq \frac{\texttt{LB}(\delta(r_j^{\triangleright}, r_j^{\triangleleft}))}{\texttt{UB}(\delta(d_i^{\triangleright}, r_j^{\triangleright})) + \texttt{LB}(\delta(r_j^{\triangleright}, r_j^{\triangleleft})) + \texttt{UB}(\delta(r_j^{\triangleleft}, d_i^{\triangleleft}))} \tag{12}$$

And we can establish a **lower bound** of $\pi(d_i, r_j)$:

$$\texttt{LB}(\pi(d_i, r_j)) = \frac{\texttt{LB}(\delta(r_j^{\triangleright}, r_j^{\triangleleft}))}{\texttt{UB}(\delta(d_i^{\triangleright}, r_j^{\triangleright})) + \texttt{LB}(\delta(r_j^{\triangleright}, r_j^{\triangleleft})) + \texttt{UB}(\delta(r_j^{\triangleleft}, d_i^{\triangleleft}))} \tag{13}$$

Example 10. *In our running example, $\pi_3 = 0.6$ for driver d_3; for $\langle d_3, r_1\rangle$, $\texttt{UB}(\pi(d_3, r_1)) = 0.75$, $\texttt{LB}(\pi(d_3, r_1)) = 0.50$. Since $\texttt{LB}(\pi(d_3, r_1)) < \pi_3 < \texttt{UB}(\pi(d_3, r_1))$, d_3 is considered as a candidate driver for r_1.*

5 Exact Method for Search-based RS

For `Search-basedRS`, given a rider r_j who posts a search request, a straightforward method first computes its candidate driver set $\tilde{\mathcal{C}}_j = D_j^{\triangleright} \cap D_j^{\triangleleft}$, then enumerates every driver d_i in $\tilde{\mathcal{C}}_j$, computes the SRP value and returns the top-k drivers with the largest SRP values. However, if there are large numbers of candidate drivers, it is expensive to compute the shortest-route distances for all candidate drivers. To address this problem, we propose an expansion-based method that progressively accesses the candidates in $\tilde{\mathcal{C}}_j$ and prunes unnecessary drivers (Section 5.1). We then

1041-4347 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2760880, IEEE Transactions on Knowledge and Data Engineering

propose a best-first method which first accesses the drivers with large probabilities in the top-k results and prunes the drivers not in the top-k results (Section 5.2).

5.1 An Expansion-Based Method

Given a rider r_j, for each driver d_i, their SRP only depends on the distance between their sources and the distance between their destinations, based on the definition of SRP ($\frac{\delta(r_j^{\triangleright}, r_j^{\triangleleft})}{\delta(d_i^{\triangleright}, r_j^{\triangleright}) + \delta(r_j^{\triangleright}, r_j^{\triangleleft}) + \delta(r_j^{\triangleleft}, d_i^{\triangleleft})}$). Thus the drivers with closer source and destination to r_j's source and destination have larger SRP values. Thus we access the drivers in $D_j^{\triangleright}$ and $D_j^{\triangleleft}$ sorted by the network distance to $r_j^{\triangleright}$ and $r_j^{\triangleleft}$ in ascending order respectively. Obviously, the first accessed drivers have large probability to be in the top-k results. Suppose $D_j^{\triangleright}[z]$ ($D_j^{\triangleleft}[z]$) denotes the z^{th} closest drivers to $r_j^{\triangleright}$ ($r_j^{\triangleleft}$).

We utilize the Dijkstra algorithm to progressively access drivers that are close to $r_j^{\triangleright}$ and $r_j^{\triangleleft}$ respectively, by expending from $r_j^{\triangleright}$ and $r_j^{\triangleleft}$. Thus we can access $D_j^{\triangleright}[z]$ and $D_j^{\triangleleft}[z]$ in an ascending order of the network distance. We use a priority queue Q to keep the current top-k results. Q_k denotes the k^{th} largest SRP value in Q. For $d = D_j^{\triangleright}[z]$, $d' = D_j^{\triangleleft}[z]$, we use $\delta(d^{\triangleright}, r_j^{\triangleright})$ and $\delta(r_j^{\triangleleft}, d'^{\triangleleft})$ to compute an SRP bound: $\frac{\delta(r_j^{\triangleright}, r_j^{\triangleleft})}{\delta(d^{\triangleright}, r_j^{\triangleright}) + \delta(r_j^{\triangleright}, r_j^{\triangleleft}) + \delta(r_j^{\triangleleft}, d'^{\triangleleft})}$, and compare it with Q_k:

(1) if the bound is smaller than Q_k, the algorithm terminates and the results in Q are the final results ($\mathcal{C}_j^k = Q$), because all the drivers after d and d' cannot be a top-k answer or has been added into Q;

(2) otherwise, we compute $\pi(d, r_j)$ and $\pi(d', r_j)$ of d and d'. If these SRP values are larger than Q_k, we use them to update Q.

Algorithm 3 shows the pseudo code of the expansion-based method for `Search-basedRS`. The initial SRP bound is computed using the driver with the closest source to r_j's source location $D_j^{\triangleright}[1]$ and the driver with the closest destination to r_j's destination $D_j^{\triangleleft}[1]$(line 3). During the expansion, for a newly computed bound value using $d = D_j^{\triangleright}[z]$ and $d' = D_j^{\triangleleft}[z]$, if the bound value is smaller than the current k^{th} largest SRP value in Q, Q_k, all the top-k matching drivers have been found, the algorithm terminates (lines 6-7); Otherwise, the shared route percentages between d and r_j, and d' and r_j are computed to update Q (line 8). Then the algorithm finds the next closest d and d' to update the SRP bound (line 9) and compares it with Q_k in the next iteration.

We can prove that $\mathcal{C}_j^k$ computed using the above expansion-based method is *correct*, that is, the top-k optimal matching drivers are contained in $\mathcal{C}_j^k$. Theorem 2 guarantees the completeness of $\mathcal{C}_j^k$.

Theorem 2. *For rider r_j, $\mathcal{C}_j^k$ generated using the expansion-based method is correct.*

Proof. We prove by contradiction and prove that when the terminating condition is met, there is no driver in $D \backslash \mathcal{C}_j^k$ with a higher SRP value than any of the drivers in $\mathcal{C}_j^k$.

Let $Q_k = d_k$, $\pi(d_k, r_j) = \frac{\delta(r_j^{\triangleright}, r_j^{\triangleleft})}{\delta(d_k^{\triangleright}, r_j^{\triangleright}) + \delta(r_j^{\triangleright}, r_j^{\triangleleft}) + \delta(r_j^{\triangleleft}, d_k^{\triangleleft})}$, and $\pi(d_{\overline{k}}, r_j) = \frac{\delta(r_j^{\triangleright}, r_j^{\triangleleft})}{\delta(d_{\overline{k}}^{\triangleright}, r_j^{\triangleright}) + \delta(r_j^{\triangleright}, r_j^{\triangleleft}) + \delta(r_j^{\triangleleft}, d_{\overline{k}}^{\triangleleft})}$, for any driver $d_{\overline{k}} \in D \backslash \mathcal{C}_j^k$.

If $d_{\overline{k}}$ was one of the top-k drivers for r_j, we should have $\pi(d_k, r_j) \leq \pi(d_{\overline{k}}, r_j)$, and $\delta(d_k^{\triangleright}, r_j^{\triangleright}) + \delta(r_j^{\triangleleft}, d_k^{\triangleleft}) \geq \delta(d_{\overline{k}}^{\triangleright}, r_j^{\triangleright}) + \delta(r_j^{\triangleleft}, d_{\overline{k}}^{\triangleleft})$, meaning that $d_{\overline{k}}$ should appear earlier than d_k in either $D_j^{\triangleright}$ or $D_j^{\triangleleft}$, which is contradictory with the real order of d_k and $d_{\overline{k}}$.

Therefore, the expansion-based method is correct. □

Algorithm 3: `Expansion`$(r_j, D, G\langle V, E\rangle)$

Input: r_j: a rider, D: the drivers set, $G\langle V, E\rangle$: the road network
Output: $\mathcal{C}_j^k$: the top-k matching drivers for r_j

1 **begin**
2 $Q = \{\}$, $z = 1$, $d = D_j^{\triangleright}[z]$, $d' = D_j^{\triangleleft}[z]$;
3 $bound = \frac{\delta(r_j^{\triangleright}, r_j^{\triangleleft})}{\delta(d^{\triangleright}, r_j^{\triangleright}) + \delta(r_j^{\triangleright}, r_j^{\triangleleft}) + \delta(r_j^{\triangleleft}, d'^{\triangleleft})}$;
4 **while** *true* **do**
5 **if** $bound < Q_k$ **then**
6 Break;
7 **else**
8 Compute $\pi(d, r_j)$, $\pi(d', r_j)$; update Q;
9 Utilize Dijkstra to compute next $d = D_j^{\triangleright}[z]$, $d' = D_j^{\triangleleft}[z]$; update *bound*;
10 $\mathcal{C}_j^k = Q$;
11 **return** $\mathcal{C}_j^k$;
12 **end**

Fig. 7. Expansion-Based Algorithm for `Search-basedRS`.

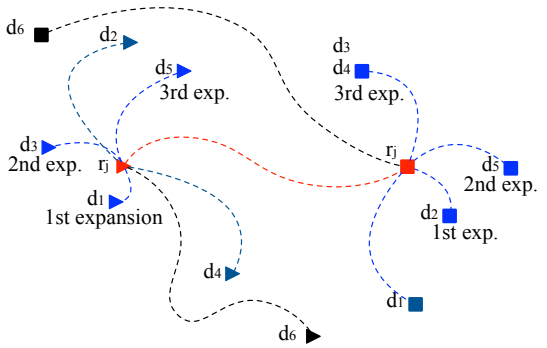


Fig. 8. Expansion-based Example for `Search-basedRS`.

Complexity. We utilize the Dijkstra algorithm to compute the next closest drivers, and the complexity is $\mathcal{O}(|E|)$. Suppose we use G-tree to compute the shortest routes, the complexity is $\mathcal{O}(\log_f \frac{|V|}{l}|V|)$. Suppose the algorithm terminates in y iterations. The complexity of the expansion-based method is $\mathcal{O}(|\mathcal{C}_j| + (|E| + \log_f \frac{|V|}{l}|V|)y)$.

Example 11. *We illustrate the expansion-based method in Figure 8, where triangles/squares are source/destination locations. Let $k = 2$. For r_j, $D_j^{\triangleright}[1] = d_1$, $D_j^{\triangleleft}[1] = d_2$, as Q is empty, we directly enqueue d_1 and d_2 into Q, $Q_1 = d_1, Q_2 = d_2$. Next, $D_j^{\triangleright}[2] = d_3$, $D_j^{\triangleleft}[2] = d_5$, the bound $\frac{\delta(r_j^{\triangleright}, r_j^{\triangleleft})}{\delta(d_3^{\triangleright}, r_j^{\triangleright}) + \delta(r_j^{\triangleright}, r_j^{\triangleleft}) + \delta(r_j^{\triangleleft}, d_5^{\triangleleft})}$ is larger than Q_2, thus we compute $\pi(d_3, r_j)$ and $\pi(d_5, r_j)$, and enqueue them to Q. Now $Q_1 = d_3, Q_2 = d_5$. $D_j^{\triangleright}[3] = d_4$, $D_j^{\triangleleft}[3] = d_4$, the new bound $\frac{\delta(r_j^{\triangleright}, r_j^{\triangleleft})}{\delta(d_4^{\triangleright}, r_j^{\triangleright}) + \delta(r_j^{\triangleright}, r_j^{\triangleleft}) + \delta(r_j^{\triangleleft}, d_4^{\triangleleft})} > Q_2$, and the expansion is terminated. Finally $\mathcal{C}_j^k = \{d_3, d_5\}$ are the top 2 matching drivers for r_j. Any other drivers, e.g., d_6, can not be a possible top-k matching, and are effectively excluded.*

5.2 A Best-First Method

The expansion-based method requires to utilize the Dijkstra algorithm in the expansion, which may be expensive. In

1041-4347 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2760880, IEEE Transactions on Knowledge and Data Engineering

this section, we propose a best-first method which utilizes bounds to identify the "best" drivers that have high probabilities in the top-k results.

Pre-processing $\tilde{\mathcal{C}}_j$. First, we generate r_j's candidate drivers set $\tilde{\mathcal{C}}_j$ using $(\frac{1}{\pi_0}-1)\delta(r_j^{\triangleright}, r_j^{\triangleleft})$ similar to the method in the `Join-basedRS` model. For each candidate driver $d_i \in \tilde{\mathcal{C}}_j$, we compute

$$\texttt{UB}(d_i) = \texttt{UB}(\delta(d_i^{\triangleright}, r_j^{\triangleright})) + \delta(r_j^{\triangleright}, r_j^{\triangleleft}) + \texttt{UB}(\delta(r_j^{\triangleleft}, d_i^{\triangleleft})) \quad (14)$$

and

$$\texttt{LB}(d_i) = \texttt{LB}(\delta(d_i^{\triangleright}, r_j^{\triangleright})) + \delta(r_j^{\triangleright}, r_j^{\triangleleft}) + \texttt{LB}(\delta(r_j^{\triangleleft}, d_i^{\triangleleft})) \quad (15)$$

where upper bounds of network distances are computed using the partition-based index in Section 4.2 and lower bounds are estimated by Euclidean distances.

Pruning. We order all the $\texttt{UB}(d_i)$ values ascendantly. Let $\texttt{UB}_k$ denote the k^{th} smallest $\texttt{UB}(d_i)$. Then we can remove the drivers(e.g., $d_{i'}$) whose lower bounds are larger than $\texttt{UB}_k$ ($\texttt{LB}(d_{i'}) > \texttt{UB}_k$).

Computing the top-k matching drivers. We use a priority queue Q to keep the current top-k drivers result. Obviously the driver with smaller lower bounds should have larger probability to be in the top-k results. Thus for each unpruned driver d_i, we access them in ascending order by $\texttt{LB}(d_i)$ and compute $\pi(d_i, r_j)$, and compare it with Q_k. If $\pi(d_i, r_j) > Q_k$, we enqueue d_i to Q to update the top-k result. Finally the results in Q are the top-k matching drivers ($\mathcal{C}_j^k = Q$).

Algorithm 4 shows the pseudo code of the best-first method for `Search-basedRS`. Function CANDIDATESETGEN first generates the set of candidate drivers $\tilde{\mathcal{C}}_j$ for rider r_j (line 2). Then the upper bounds and lower bounds of total traveled network distances by drivers in $\tilde{\mathcal{C}}_j$ are computed (line 3). Any driver d whose lower bound of total traveling network distance for sharing ride with rider r_j that is larger than the k^{th} smallest upper bound of the total network distance traveled by another driver d' can not be the top-k matching drivers, they can be safely pruned (line 4). For each unpruned driver d_i, if the lower bound of d_i's traveled network distance is larger than the total distance traveled by the driver of Q_k, d_i and drivers with larger lower bounds of total traveled network distance can not be the top-k matching drivers, they can be safely pruned (lines 6-7). Otherwise, if d_i is a valid driver for r_j and the SRP value $\pi(d_i, r_j)$ is better than Q_k, d_i is enqueued to Q (line 10) to produce the final result (line 12).

Complexity. The computations of $\texttt{UB}(d_i)$ and $\texttt{LB}(d_i)$ take $\mathcal{O}(1)$ time. Suppose we use G-tree to compute the shortest routes, the complexity is $\mathcal{O}(\log_f \frac{|V|}{l}|V|)$. Suppose the algorithm terminates in y iterations. The complexity of the best-first method is $\mathcal{O}(|\mathcal{C}_j| + \log_f \frac{|V|}{l}|V|y)$.

Example 12. *Suppose we want to find the top-3 drivers from $\mathcal{C}_j = \{d_1, d_2, d_3, d_4, d_5\}$ for a rider r_j. The upper bounds $\texttt{UB}(d_i)$ and lower bounds $\texttt{LB}(d_i)$ for $1 \le i \le 5$ are: $13, 15, 10, 11, 19$, and $12, 14, 9, 7, 16$. $\texttt{UB}_3 = \texttt{UB}(d_1) = 13$. For d_2 and d_5, $\texttt{LB}(d_2) = 14 > \texttt{UB}_3$, $\texttt{LB}(d_5) = 16 > \texttt{UB}_3$, thus d_2 and d_5 cannot be one of the top-3 drivers, they are discarded ($\pi(d_2, r_j) = 13.2$, $\pi(d_5, r_j) = 17$). The remaining drivers are d_1, d_3 and d_4. We compute their exact SRP values and enqueue them to Q. Therefore, the result $\mathcal{C}_j^k = \{d_1, d_3, d_4\}$.*

Algorithm 4: `Best-First`$(r_j, D, G\langle V, E\rangle)$

Input: r_j: a rider, D: the drivers set, $G\langle V, E\rangle$: the road network
Output: $\mathcal{C}_j^k$: the top-k matching drivers for r_j

```
1  begin
2      C̃_j = CandidateSetGen({r_j}, D, G⟨V,E⟩, π_0);
3      ∀d_i ∈ C̃_j, compute UB(d_i) and LB(d_i);
4      Prune all drivers d_i' having LB(d_i') > UB_k;
5      for each unpruned d_i by LB(d_i) ascending order do
6          if LB(d_i) >total distance by driver Q_k then
7              break;
8          else
9              if π(d_i, r_j) ≥ Q_k and π(d_i, r_j) ≥ π_i then
10                 Enqueue π(d_i, r_j) to Q;
11     C_j^k = Q;
12     return C_j^k;
13 end
```

Fig. 9. Best-First Algorithm for `Search-basedRS`.

TABLE 2
Test Data Sets

Data Set	Num. of Trajectories	Average Trajectory Length (km)
`Taxi`	200,000	6.128
`UCar`	1,267,000	11.607

6 EXPERIMENTS

In this section, we conduct experiments on real datasets to evaluate the performance of algorithms and the quality of our approximate algorithms.

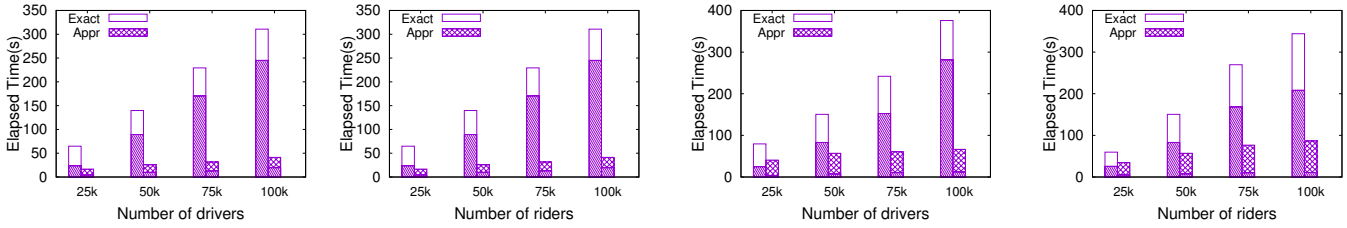
Datasets. We use a real road network of Beijing with 338,024 vertices and 440,525 edges. We use two ride-sharing datasets, `Taxi` (www.datatang.com/data/45888) and `UCar` (www.zhuanche.zuche.com, a commercial service like Uber). `Taxi` contains about $200,000$ trajectories of user orders generated by more than 8,000 public taxicabs in one month in Beijing. `UCar` contains $1,267,000$ trajectories of user orders generated by nearly 2,000 private drivers within one week in Beijing. We extract the pick-up and drop-off locations from `Taxi` and `UCar` and randomly assign them to a set of drivers and a set of riders. All tests are performed separately on the drivers/riders sets generated from `Taxi` and `UCar`. We use normal distribution to model the distribution of the SRP requirement (π_i) of each driver, and $\pi_0 \le \pi_i \le 1$. First, given π_0, the average of the normal distribution of random variable x is set to π_0, and the variance is $1 - \pi_0$. Then we use normal distribution to generate a value of x. If $x \le 1$, we set $\pi_i = x$; If $x > 1$, we set $\pi_i = 1$. We also tested other distributions of π_i, such as uniform distribution, and our experimental results indicated that the distribution of π_i had little impact on the performance of our algorithms on efficiency and effectiveness. Besides, normal distribution was more practical for such scenario, and thus we used normal distribution of π_i.

Experimental Setting. All of the algorithms are implemented in C++ with -O3 flag. All the experiments are conducted in a machine with 2.40 GHz Intel Xeon CPU E5-2630, 48 GB RAM, running Ubuntu 14.04.

Algorithms. Table 3 summarises the complexities of our proposed algorithms. The approximate guarantee of the approximate algorithm for `Join-basedRS` is τ.

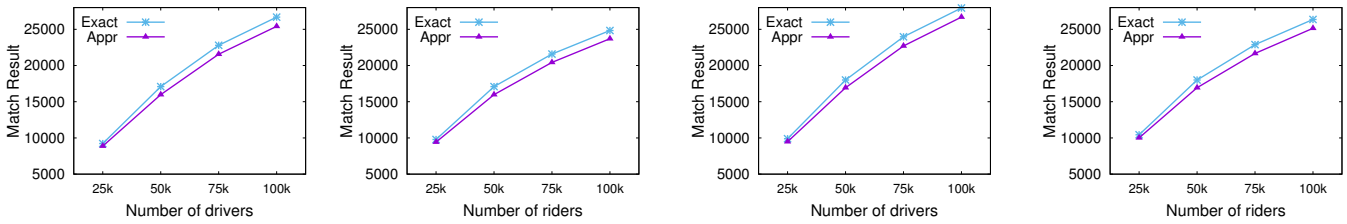
1041-4347 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2760880, IEEE Transactions on Knowledge and Data Engineering



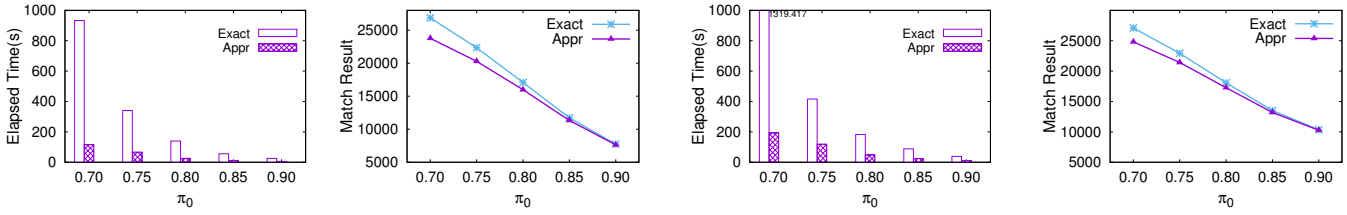
(a) Taxi, 50000 riders (b) Taxi, 50000 drivers (c) UCar, 50000 riders (d) UCar, 50000 drivers

Fig. 10. Test1 - Join-basedRS Efficiency by Varying # of Riders/Drivers ($\pi_0 = 0.8$, $\tau = 1.5$, $h = 100$).



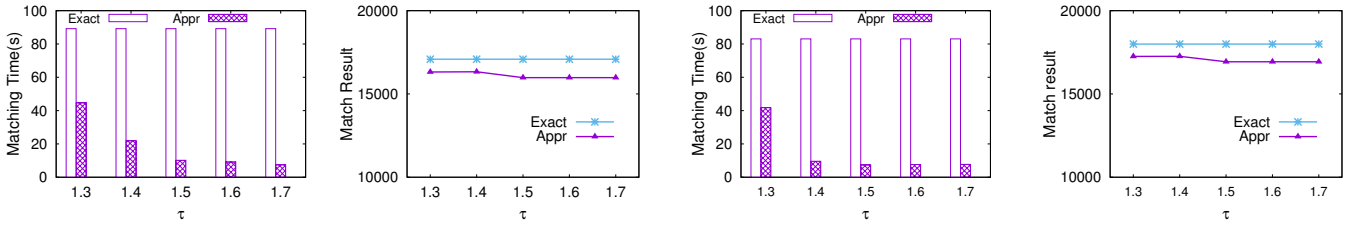
(a) Taxi, 50000 riders (b) Taxi, 50000 drivers (c) UCar, 50000 riders (d) UCar, 50000 drivers

Fig. 11. Test2 - Join-basedRS Effectiveness by Varying # of Riders/Drivers ($\pi_0 = 0.8$, $\tau = 1.5$, $h = 100$).



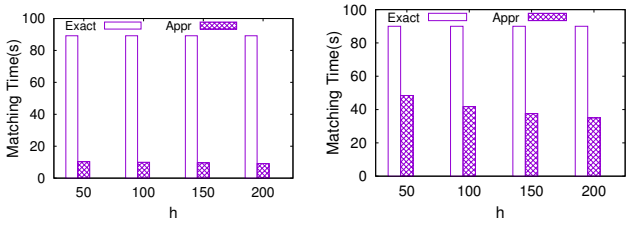
(a) Taxi (b) Taxi (c) UCar (d) UCar

Fig. 12. Test3 - Join-basedRS by Varying π_0 (50000 drivers, 50000 riders, $\tau = 1.5$, $h = 100$).



(a) Taxi. Efficiency (b) Taxi. Effectiveness (c) UCar. Efficiency (d) UCar. Effectiveness

Fig. 13. Test4 - Join-basedRS by Varying τ (50000 drivers, 50000 riders, $\pi_0 = 0.8$, $h = 100$).



(a) Taxi (b) UCar

Fig. 14. Test5 - Join-basedRS efficiency by Varying h (50000 drivers, 50000 riders, $\pi_0 = 0.8, \tau = 1.5$).

6.1 Evaluating Join-based RS

In this section, we evaluate the Join-basedRS techniques. We implemented two strategies: (1) Exact: the exact method in Section 3 that uses exact shortest routes of riders to generate candidate driver-rider set and uses bigraph matching algorithms to compute the precise matching P^*. (2) Appr: the approximate solution in Section 4 that uses bounds of riders' shortest routes to construct the upper and lower bound bigraphs $\mathcal{G}^U$ and $\mathcal{G}^L$, and uses the converged $\mathtt{LB}(P)$ as the approximate answer for P^*. We evaluate the efficiency, effectiveness, and their trends by varying π_0 (the minimal shared route percentage by all drivers), τ (the bound ratio threshold for $\mathtt{UB}(P)/\mathtt{LB}(P)$ in Appr iteration) and h (the number of upper/lower bound bigraph edges to update into exact shared percentages in each iteration of Appr). By default, $\pi_0 = 0.8$, $\tau = 1.5$ and $h = 100$ for our Join-basedRS algorithms, and $k = 3$, $\pi_0 = 0.7$ for the Search-basedRS algorithms. We use the G-tree algorithm when computing shortest routes.

6.1.1 Test1: Efficiency

We evaluate the efficiency by fixing the number of riders and varying the number of drivers, and vice versa. The running time is composed of two parts: candidate set generation (i.e. bigraph construction) and bigraph matching (for Appr the matching time is the converging time for making $\frac{\mathtt{UB}(P)}{\mathtt{LB}(P)} \leq \tau$).

1041-4347 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2760880, IEEE Transactions on Knowledge and Data Engineering

TABLE 3
Algorithm Complexity (y is the iteration times before an algorithm terminates, h is the number of edges to update to real SRP)

RS Problem	Method	Complexity
Join-based	Exact	$\mathcal{O}(\lvert\tilde{\mathcal{C}}'\rvert + \log_f \frac{\lvert V\rvert}{l}\lvert V\rvert\lvert\tilde{\mathcal{C}}\rvert + (\lvert R\rvert+\lvert D\rvert)^2\lvert\tilde{\mathcal{C}}\rvert)$
	Approximate	$\mathcal{O}(yh\log_f \frac{\lvert V\rvert}{l}\lvert V\rvert + y(\lvert R\rvert+\lvert D\rvert))$
Search-based	Baseline	$\mathcal{O}(\log_f \frac{\lvert V\rvert}{l}\lvert V\rvert\lvert\mathcal{C}_j\rvert)$
	Expansion	$\mathcal{O}(\lvert\mathcal{C}_j\rvert + (\lvert E\rvert + \log_f \frac{\lvert V\rvert}{l}\lvert V\rvert)y)$
	Best-first	$\mathcal{O}(\lvert\mathcal{C}_j\rvert + \log_f \frac{\lvert V\rvert}{l}\lvert V\rvert y)$

Figures 10(a)-10(b) show the results on Taxi dataset and Figures 10(c)-10(d) show the results on the UCar dataset. Each bar in the figures corresponds to a total running time of a compared strategy for certain number of drivers/riders. The upper part of a bar is the bigraph construction time, and the lower part of a bar is the bigraph matching time. The driver grouping optimazation is applied in the reported results, and on average its efficiency improvement increases linearly as the total number of drivers increases.

We have the following observations. Firstly, Appr effectively improves the overall efficiency, it runs 6 times and 5 times faster than Exact for Taxi and UCar respectively. The efficiency improvement is achieved by avoiding shortest route computations in the bigraph construction step, and the upper/lower bounds converging technique in the bigraph matching step. Secondly, with the increasing of dataset size, the efficiency improvement by Appr becomes more significant, because more shortest routes and SRP calculations are avoided, demonstrating ability to scale to large dataset. Thirdly, the candidate set generation time takes larger part in the overall running time on the UCar dataset than on the Taxi dataset. This is because orders in the UCar dataset has longer route distances and larger searching area for drivers, and thus a single rider has more candidate drivers than in the Taxi dataset and takes the higher ratio of candidate set generation time in the overall running time. Fourthly, the matching time is greatly reduced by Appr compared with that in Exact, because the expensive exact bigraph matching process is eliminated.

In all, Appr demonstrates notable ability to improve the efficiency for Join-basedRS.

6.1.2 Test2: Effectiveness

We use the summary of all shared route percentage of matched driver-rider pairs as matching results to indicate the effectiveness. The closer $\texttt{LB}(P)$ is to P^*, the higher the approximate ratio, the better the approximation methods. Figures 11(a)-11(b) show the results on Taxi dataset and Figures 11(c)-11(d) show the results on the UCar dataset, where the y-axis "Match Result" is the total shared route percentage of each matched bigraph.

We have the following observations. Firstly, the approximate answers by Appr are within 93.5%-96.3% and 94.1%-95.9% to the exact answer on Taxi and UCar respectively. The approximating ability is attributable to bound convergence technique of Appr. Secondly, the approximate answers for smaller datasets are closer to the precise answer because larger portion of bigraph edges are updated to exact SRP values in each upper/lower bound convergence iteration. Thirdly, the approximate rate, i.e., $\frac{P^*}{\texttt{LB}(P)}$, is always bounded within the predefined threshold τ, and are much smaller as illustrated by our experimental results.

In all, the approximate answers by Appr are close enough to accurate answers and can be achieved faster.

6.1.3 Test3: Impact of the minimal SRP value π_0

We evaluate how the global shared route percentage constraint π_0 affects the overall matching result and the running time. Figures 12(a)-12(b) show the results on Taxi dataset and Figures 12(c)-12(d) show the results on the UCar dataset.

We have the following observations. Firstly, as π_0 increases, the running time of both Exact and Appr decreases. This is because larger π_0 constraints for drivers correspond to higher selectivity, i.e., fewer candidate drivers for each rider and less bigraph edges, and therefore faster candidate generation and matching. Secondly, the improvement of runtime efficiency by Appr over Exact is more obvious with smaller π_0 values than with larger π_0 values, where the selectivity of π_0 contributes less to improving efficiency, demonstrating the speeding power of Appr. Thirdly, as π_0 increases, the approximate rate increases for the Taxi dataset. This is because edge numbers are reduced in both upper bound bigraph and lower bound bigraph, resulting in a smaller gap between $\texttt{UB}(P)$ and $\texttt{LB}(P)$, and closer converged $\texttt{LB}(P)$ to precise answer P^*. In the meantime, the overall matching value is reduced as π_0 increases, indicating that the π_0 value should be set prudently to achieve overall requirement.

In all, Appr can achieve high performance (less than 100 seconds) for different π and can be used in practice.

6.1.4 Test4: Impact of the graph bound threshold τ

We evaluate the trends of matching time and approximate ratio of Appr versus Exact as τ varies. Figures 13(a)-13(b) and Figures 13(c)-13(d) show the results on the Taxi and UCar datasets.

We have the following observations. Firstly, the matching time of Appr keeps decreasing as τ increases, while Exact is not affected as it uses exact bigraph matching technique. This is because larger τ means larger tolerance on the approximate answer, and less convergence time for $\texttt{UB}(P)/\texttt{LB}(P)$ to approach τ. Secondly, the approximate ratio decreases as τ increases, and the decreasing trend is more obvious for $\tau \leq 1.5$ than for $\tau > 1.5$. Again, this is attributable to the approximation tolerance introduced by different τ values.

In all, for Appr, for τ between 1 and 2, the matching quality is similar, around 95%. A larger threshold leads to higher performance.

6.1.5 Test5: Impact of the edge numbers h between iterations for Appr

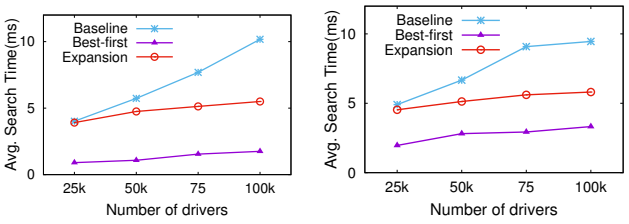
We evaluate how the h parameter affects the convergence of Appr. Figure 14(a) and Figure 14(b) show the results on the Taxi and UCar datasets.

We observe that the more edges are updated into exact SRP values in both the upper bound and lower bound bigraphs, the faster that $\frac{\texttt{UB}(P)}{\texttt{LB}(P)}$ converges to τ. The efficiency improvement of Appr over Exact is more obvious on the Taxi dataset than on the UCar dataset because the average network distance for UCar orders is longer than that of the Taxi orders, which results in larger candidate sets, and thus larger bigraphs and longer matching time.

We do not evaluate the quality because different h values will not affect the quality.

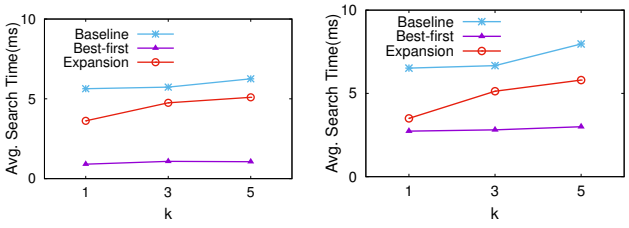
1041-4347 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2760880, IEEE Transactions on Knowledge and Data Engineering



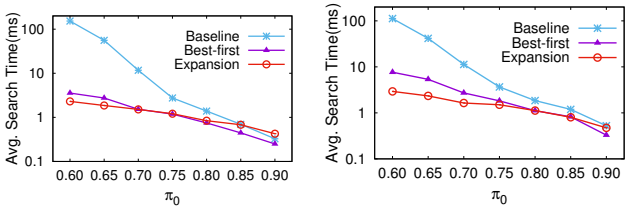
(a) Taxi (b) UCar

Fig. 15. Test6 - Search-basedRS Efficiency by Varying # of Drivers ($\pi_0 = 0.7$, $k = 3$).



(a) Taxi (b) UCar

Fig. 16. Test7 - Search-basedRS Efficiency by Varying k (50000 drivers, $\pi_0 = 0.7$).



(a) Taxi (b) UCar

Fig. 17. Test8 - Search-basedRS Efficiency by Varying π_0 (100000 drivers, $k = 3$).

6.2 Evaluating Search-based RS

In this section, we evaluate Search-basedRS techniques. We implement three strategies: (1) Baseline: the baseline algorithm that searches all possible candidate drivers, computes all SRP values and selects the top k drivers for a given rider. (2) Expansion: Algorithm 3 in Section 5, the expansion-based method that gradually increases the candidate drivers set for a single passenger. (3) Best-first: Algorithm 4 in Section 5 that only computes k driver-rider shared route percentages in the best scenario. We conduct three groups of tests to evaluate the impact of efficiency by varying one of the following three parameters while fixing the other two: number of drivers, k and π_0.

6.2.1 Test6: Impact on efficiency by number of drivers

We evaluate the runtime efficiency by varying the number of drivers ($\pi_0 = 0.7$, $k = 3$), and . Figure 15(a) and Figure 15(b) show the results on the Taxi and UCar datasets.

We have the following observations. Firstly, Best-first has the highest efficiency, and Expansion runs faster than Baseline. Best-first runs 6 times faster than Baseline on the Taxi dataset, and 3 times faster on the UCar dataset. This is because Best-first does not need to compute all the $\pi(d_i, r_j)$ values for each $d_i \in \mathcal{C}_j$ of the single rider r_j, drivers whose lower bounds $\mathtt{LB}(d_i)$ are larger than the k^{th} biggest SRP value ($Q(k)$) so far are directly discarded. Expansion beats Baseline because the expansion stops when the size of the intersection of the candidate driver list for $r_j^{\triangleright}$ and the candidate driver list for $r_j^{\triangleleft}$ reaches k, thus avoiding calculation of SRP values for all the excluded drivers. Best-first outperforms Expansion because the expansion process is more expensive than the bound-based method adopted by Best-first. Secondly, the efficiency improvements by Best-first and Expansion are more obvious on the Taxi dataset than on the UCar dataset. For Best-first this is because on average the UCar dataset has larger candidate searching area and thus more candidate drivers, and accordingly, the portion of drivers whose lower bounds $\mathtt{LB}(d_i)$ are larger than $Q(k)$ in the whole $\mathcal{C}_j$ set is lower than that of the Taxi dataset, introducing more exact SRP computations and thus higher average running time. The reason for Expansion is similar, larger candidate searching area introduces more must-check candidate drivers. Thirdly, both Best-first and Expansion have better scalability than Baseline. As the number of available drivers increases, the time consumption of Best-first and Expansion increases slower than Baseline. The reason is that Baseline always computes all exact SRP values for all the valid candidate drivers, while the pruning power of Best-first and Expansion becomes more obvious, i.e., the more avoidance of exact SRP computations, the more time reduction.

In addition, the answers returned by Best-first and Expansion methods are always consistent with that of Baseline, the exact solution, demonstrating the correctness.

6.2.2 Test7: Impact on efficiency by k

We evaluate the search performance by varying k ($\pi_0 = 0.7$, 50000 drivers). Figure 16(a) and Figure 16(b) show the results on the Taxi and UCar datasets.

We have the following observations. Firstly, for each k value, Best-first runs faster than Baseline and Expansion, because of the pruning power analyzed in Test6. Secondly, as k increases, the runtime increasing rates of Best-first and Expansion are much slower than Baseline, which is also attributed to the effective pruning power.

6.2.3 Test8: Impact on efficiency by π_0

We evaluate the search performance by varying π_0 ($k = 3$, 100000 drivers). Figure 17(a) and Figure 17(b) show the results on the Taxi and UCar datasets.

We have the following observations. Firstly, for most π_0 values, Best-first and Expansion always run faster than Baseline, because of the effective pruning techniques of Best-first and Expansion as analyzed in Test6. Secondly, as π_0 increases, the average runtime for all three strategies decrease because of increased selectivity of larger π_0 values and less number of candidates. Thirdly, the performance improvement of Best-first and Expansion over Baseline is more obvious for smaller π_0 values (i.e., $\pi_0 \leq 0.85$) than for larger π_0 values, because smaller π_0 values correspond to larger searching areas of drivers, thus achieving better pruning powers of Best-first and Expansion.

In all, the Best-first method is the most efficient and scalable technique for Search-basedRS.

7 Related Works

7.1 Ride-sharing

The ride-sharing problem has gained increasing attention in both the academic and the industrial communities[3], [5],

1041-4347 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2760880, IEEE Transactions on Knowledge and Data Engineering

[10], [14], [19], [20], [26], [27], where the goal is to match drivers and riders on a real-time basis, within the riders' time windows and drivers' capacities[5], [14], [19], [26], [27].

In the auction-based framework of [5], drivers can automatically bid on nearby riders' requests, so that the server assigns each rider to the highest bidder(driver), taking into consideration both the rider's and the driver's constraints, such as pricing model, vehicle capacity, profits, etc. In [14], ride requests are prioritized according to their expiration time. Heuristics such as saving gains and greedy grouping are used to group ride requests and assign them to certain drivers. The grouping problem is solved by continuous stream query and extended to parallel algorithm to handle large-scale on-line ride requests. [26] studies a similar ride-sharing problem, focusing on the satisfaction rate and approximate routes, among which one greedy route is provided to meet the ride request. In [20], methods to generate shared transportation plans are reviewed, and the problem is then deemed as an agent collaboration model, where multiple agents and market-based incentives are coordinated for the key challenges. In [10] a distributed taxi-sharing system is proposed, and empirically studied. The service quality is similar to traditional taxis. In [19], to provide fast response time, trip requests with waiting and service time constraints are served in a sequential style, i.e., the requests are assigned to drivers one by one, such that for each driver, the total traveling time of the ride-sharing schedule is bounded in a set range versus the traveling time without sharing. Valid schedule plans for each driver are kept in a kinetic tree structure. Given a new ride request, the matching vehicle is the one with the minimum traveling cost increase. The matching mechanism is local optimal as a matched rider is not available to be matched to other drivers anymore. In [27], riders and drivers arrive and leave independently. For each sharing schedule, riders pay no more than without sharing and drivers earn no less than not sharing. Given a new ride request, the best driver is assigned with the time and monetary constraints satisfied. The algorithm can be extended to handle large-scale ride requests, while it is not necessary to reorder schedules, as the trade-off between increased time consumption and the effectiveness is unworthy.

Our solution is different from above works. Firstly, the contexts are different. Most existing works study how to allocate vehicle capacities between riders, ignoring the drivers' own travel needs; in contrast, we acknowledge that both riders and drivers have travel needs, and study two types of such ride-sharing problems to fit into different circumstances. Secondly, we recognize the economy of sharing, and propose algorithms to maximize the overall resource utilization. Thirdly, our algorithms can properly handle large-scale ride-sharing problems, where we use the bigraph matching and the top-k models and their approximations to achieve efficiency.

7.2 Spatial Crowdsourcing

Ride-sharing can be considered as one of spatial crowdsourcing applications, where spatial task assignment is a closely-related topic. For static scenarios, the assignment problem can be regarded as the classical maximum weighted bipartite graph matching problem [22], while this is not suitable for dynamic scenarios where tasks and workers appear dynamically and the whole bigraph cannot be known in advance. Tong et al [30] experimentally study the online minimum bipartite matching in real time spatial data (OMBM) problem, where workers are predefined and tasks arrive on a real-time basis.

In [31], the Global Online Micro-task Allocation in spatial crowdsourcing (GOMA) problem is identified, and a two-phase framework with $\frac{1}{4}$-competitive ratio assuming random order model is proposed, to address the need to immediately match tasks and workers upon their arrival.

The problems in our paper are different from problems in above works, as each spatial task (rider request) in our settings has two related locations (source and destination) to be considered during the driver/rider matching process, while the tasks of [30] and [31] are characterized by one location. Moreover, we consider the road-network distance.

7.3 Route Recommendation

Route recommendation is a related topic to ride-sharing, drivers can adopt the recommended routes instead of the shortest routes. We review general-purpose route recommendation [7], [8], [16], [34], [17], [25], [32], [23] and recommender systems for taxis [13], [28], [29], [35], [36], [37].

Popular routes. In [8], path desirability is evaluated by a popularity function using historical trajectories. The recommended popular routes usually have fewer drivers than routes provided by other algorithms. In [32], k popular routes are provided by mining trajectory datasets. The recommendation routes are of coarse granularity, rather than exact routes. [7] argues that system-recommended routes may be different from users' preference. Therefore, personalized popular routes are mined from the user's own historical trajectory data, to reflect differentiated preferences. In [25], the most frequent paths in different time periods are presented to users according to use-specified time period.

Recommendation for taxi dispatching. Coarse-grained recommender systems [29], [35], [24], [18], [37] suggest driving directions for drivers to pick up possible riders. In [15], [13], [36], multiple drivers take turns to get the recommended routes. A fair driver recommendation model is proposed in [28], the goal is to assign routes to a group of competing drivers on a fair basis. The driving cost per customer for each taxi is kept constant to ensure the fairness. Each candidate route is scored according to the probability of successful custom pick-ups along this route, and a cost matrix is assembled to fairly assign routes.

8 Conclusion

In this paper, we define and study two types of ride-sharing problem and propose an efficient framework to address the two problems. We first propose an exact algorithm for the `Join-basedRS` which can remove a large number of invalid pairs and avoid computing their shortest routes. We then propose an approximate algorithm that estimates an upper bound and a lower bound, and approaches the two bounds by computing the shortest routes for some rider-driver pairs. For `Search-basedRS`, we propose a best-first algorithm to compute the top-k results. Experimental study on two real-world datasets demonstrates the effectiveness and efficiency of our algorithms. In the future, we want to extend our method to support dynamic settings where drivers and riders are dynamically coming.

Acknowledgment. This work was supported by the 973 Program of China (2015CB358700), NSF of China

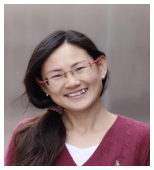
1041-4347 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2760880, IEEE Transactions on Knowledge and Data Engineering

(61632016, 61373024, 61602488, 61422205, 61472198), ARC DP170102726, and FDCT/007/2016/AFJ. Guoliang Li is the corresponding author.

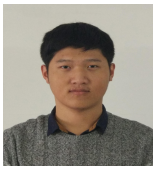
REFERENCES

[1] www.airbnb.com.
[2] www.didi.xin.
[3] www.uber.com.
[4] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows - theory, algorithms and applications*. Prentice Hall, 1993.
[5] M. Asghari, D. Deng, C. Shahabi, U. Demiryurek, and Y. Li. Price-aware real-time ride-sharing at scale: an auction-based approach. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS 2016, Burlingame, California, USA, October 31 - November 3, 2016*, pages 3:1–3:10, 2016.
[6] R. Belk. You are what you can access: Sharing and collaborative consumption online. *Journal of Business Research*, 67(8):1595–1600, 2014.
[7] V. Ceikute and C. S. Jensen. Routing service quality - local driver behavior versus routing services. In *MDM*, pages 97–106, 2013.
[8] Z. Chen, H. T. Shen, and X. Zhou. Discovering popular routes from trajectories. In *ICDE*, pages 900–911, 2011.
[9] B. Cohen and J. Kietzmann. Ride on mobility business models for the sharing economy. *Organization & Environment*, 27(3):279–296, 2014.
[10] P. M. d'Orey, R. Fernandes, and M. Ferreira. Empirical evaluation of a dynamic and distributed taxi-sharing system. In *IEEE Conference on Intelligent Transportation Systems*, pages 140–146, Sept 2012.
[11] R. Duan and S. Pettie. Linear-time approximation for maximum weight matching. *Journal of the ACM (JACM)*, 61(1):1, 2014.
[12] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.
[13] Y. Ge, H. Xiong, A. Tuzhilin, K. Xiao, M. Gruteser, and M. J. Pazzani. An energy-efficient mobile recommender system. In *SIGKDD*, pages 899–908, 2010.
[14] G. Gidófalvi, T. B. Pedersen, T. Risch, and E. Zeitler. Highly scalable trip grouping for large-scale collective transportation systems. In *EDBT*, pages 678–689, 2008.
[15] H. Hu, G. Li, Z. Bao, J. Feng, Y. Wu, Z. Gong, and Y. Xu. Top-k spatio-textual similarity join. *IEEE Trans. Knowl. Data Eng.*, 28(2):551–565, 2016.
[16] H. Hu, Y. Liu, G. Li, J. Feng, and K. Tan. A location-aware publish/subscribe framework for parameterized spatio-textual subscriptions. In *ICDE*, pages 711–722, 2015.
[17] H. Hu, Y. Zheng, Z. Bao, G. Li, J. Feng, and R. Cheng. Crowdsourced POI labelling: Location-aware result inference and task assignment. In *ICDE*, pages 61–72, 2016.
[18] W. Huang, G. Li, K. Tan, and J. Feng. Efficient safe-region construction for moving top-k spatial keyword queries. In *CIKM*, pages 932–941, 2012.
[19] Y. Huang, F. Bastani, R. Jin, and X. S. Wang. Large scale real-time ridesharing with service guarantee on road networks. *VLDB*, 7(14):2017–2028, 2014.
[20] E. Kamar and E. Horvitz. Collaboration and shared plans in the open world: Studies of ridesharing. In *IJCAI*, page 187, 2009.
[21] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.
[22] L. Kazemi and C. Shahabi. Geocrowd: enabling query answering with spatial crowdsourcing. In *SIGSPATIAL 2012 International Conference on Advances in Geographic Information Systems (formerly known as GIS), SIGSPATIAL'12, Redondo Beach, CA, USA, November 7-9, 2012*, pages 189–198, 2012.
[23] G. Li, J. Feng, and J. Xu. DESKS: direction-aware spatial keyword search. In *ICDE*, pages 474–485, 2012.
[24] Y. Liu, H. Wang, G. Li, J. Gao, H. Hu, and W. Li. ELAN: an efficient location-aware analytics system. *Big Data Research*, 5:16–21, 2016.
[25] W. Luo, H. Tan, L. Chen, and L. M. Ni. Finding time period-based most frequent path in big trajectory data. In *SIGMOD*, pages 713–724, 2013.
[26] S. Ma, Y. Zheng, and O. Wolfson. T-share: A large-scale dynamic taxi ridesharing service. In *ICDE*, pages 410–421, 2013.
[27] S. Ma, Y. Zheng, and O. Wolfson. Real-time city-scale taxi ridesharing. *IEEE Transactions on Knowledge and Data Engineering*, 27(7):1782–1795, 2015.
[28] S. Qian, J. Cao, F. L. Mouël, I. Sahel, and M. Li. SCRAM: A sharing considered route assignment mechanism for fair taxi route recommendations. In *SIGKDD*, pages 955–964, 2015.
[29] S. Qian, Y. Zhu, and M. Li. Smart recommendation by mining large-scale GPS traces. In *WCNC*, pages 3267–3272, 2012.
[30] Y. Tong, J. She, B. Ding, L. Chen, T. Wo, and K. Xu. Online minimum matching in real-time spatial data: Experiments and analysis. *PVLDB*, 9(12):1053–1064, 2016.
[31] Y. Tong, J. She, B. Ding, L. Wang, and L. Chen. Online mobile micro-task allocation in spatial crowdsourcing. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*, pages 49–60, 2016.
[32] L. Wei, Y. Zheng, and W. Peng. Constructing popular routes from uncertain trajectories. In *KDD*, pages 195–203, 2012.
[33] D. B. West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.
[34] M. Yu, G. Li, T. Wang, J. Feng, and Z. Gong. Efficient filtering algorithms for location-aware publish/subscribe. *IEEE Trans. Knowl. Data Eng.*, 27(4):950–963, 2015.
[35] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang. T-drive: driving directions based on taxi trajectories. In *GIS*, pages 99–108, 2010.
[36] N. J. Yuan, Y. Zheng, L. Zhang, and X. Xie. T-finder: A recommender system for finding passengers and vacant taxis. *IEEE Trans. Knowl. Data Eng.*, 25(10):2390–2403, 2013.
[37] M. Zhang, J. Liu, Y. Liu, Z. Hu, and L. Yi. Recommending pick-up points for taxi-drivers based on spatio-temporal clustering. In *CGC*, pages 67–72, 2012.
[38] R. Zhong, G. Li, K. Tan, L. Zhou, and Z. Gong. G-tree: An efficient and scalable index for spatial search on road networks. *IEEE Trans. Knowl. Data Eng.*, 27(8):2175–2189, 2015.



Na Ta is currently working as an assistant professor in the School of Journalism and Communication, Renmin University of China, Beijing, China. She received her PhD degree in Computer Science from Tsinghua University, Beijing, China in 2017. Her research interests mainly include computational communication and urban computing.

Guoliang Li is currently working as an associate professor in the Department of Computer Science, Tsinghua University, Beijing, China. He received his PhD degree in Computer Science from Tsinghua University, Beijing, China in 2009. His research interests mainly include data cleaning and integration and crowdsourcing.



Tianyu Zhao is currently a Ph.D student in the Department of Computer Science, Tsinghua University. He received his B.E. degree in Software Engineering from Beihang University in 2017. His research interests mainly include urban computing and spatial databases.

Jianhua Feng received his B.S., M.S. and PhD degrees in Computer Science from Tsinghua University. He is currently working as a professor of Department Computer Science in Tsinghua University. His main research interests include large-scale data management.

Hanchao Ma is with the Department of Electronic Engineering and Computer Science, Washington State University. He received his M.S. degree in Information Science from the University of Pittsburgh in 2014. His research interests mainly include spatial databases.

Zhiguo Gong is an associate professor in Faculty of Science and Technology, University of Macau. He obtained his PhD degree in Department of Computer Science, Institute of Mathematics, Chinese Academy of Science, China. His research fields include database systems and Web mining.

1041-4347 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.