Highly Efficient and Scalable Multi-hop Ride-sharing

Yixin Xu, Lars Kulik, Renata Borovica-Gajic, Abdullah Aldwyish, Jianzhong Qi
yixinx3@student.unimelb.edu.au,lkulik@unimelb.edu.au,renata.borovica@unimelb.edu.au,
aldwyish@student.unimelb.edu.au,jianzhong.qi@unimelb.edu.au
The University of Melbourne, Melbourne, Australia

ABSTRACT

On-demand ride-sharing services such as Uber and Lyft have gained tremendous popularity over the past decade, largely driven by the omnipresence of mobile devices. Ride-sharing services can provide economic and environmental benefits such as reducing traffic congestion and vehicle emissions. *Multi-hop ride-sharing* enables passengers to transfer between vehicles within a single trip, which significantly extends the benefits of ride-sharing and provides ride opportunities that are not possible otherwise. Despite its advantages, offering real-time multi-hop ride-sharing services at large scale is a challenging computational task due to the large combination of vehicles and passenger transfer points. To address these challenges, we propose exact and approximation algorithms that are scalable and achieve real-time responses for highly dynamic ride-sharing scenarios in large metropolitan areas. Our experiments on real-world datasets show the benefits of multi-hop ride-sharing services and demonstrate that our proposed algorithms are more than two orders of magnitude faster than the state-of-the-art. Our approximation algorithms offer a comparable trip quality to our exact algorithm, while improving the ride-sharing request matching time by another order of magnitude.

CCS CONCEPTS

• **Information systems** → **Location based services**; • **Applied computing** → *Transportation.*

KEYWORDS

Real-time, road networks, ride-sharing, multi-hop.

ACM Reference Format:
Yixin Xu, Lars Kulik, Renata Borovica-Gajic, Abdullah Aldwyish, Jianzhong Qi. 2020. Highly Efficient and Scalable Multi-hop Ride-sharing. In *Proceedings of SIGSPATIAL (Conference'20).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Ride-sharing has been widely adopted for on-demand transportation services as it offers more affordable trips by allowing passengers with similar routes to share the use of vehicles. Ride-sharing creates economic and environmental benefits due to the higher occupancy rates of vehicles and reduced overall travel distances.

Conference'20, November 2020, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

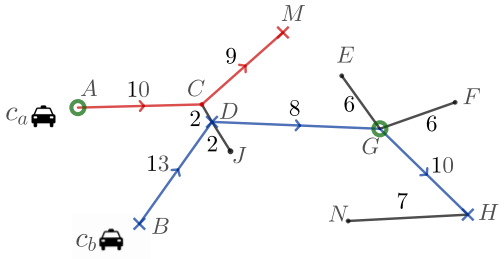


Figure 1: A multi-hop ride-sharing example.

A fundamental problem in ride-sharing is determining how to dispatch vehicles to trip requests. Dispatching (i.e., request matching) algorithms have been proposed to improve the efficiency and effectiveness of ride-sharing services [1–8]. However, most of them only consider direct trips but do not investigate the possibility of multi-hop trips. A multi-hop trip dispatches more than one vehicle to serve a request and the passengers will transfer between the dispatched vehicles. The benefits of enabling multi-hop trips have been documented in many studies [9–12]. The proposed approaches provide more flexible trips, leading to an increased matching ratio, lower travel costs of vehicles, and reduced congestion.

Figure 1 shows an example road network that denotes the travel times (in minutes) at the edges. A passenger requests a trip from A to G and needs to arrive within 25 minutes at their destination. Two vehicles c_a and c_b offer shared trips and their scheduled routes are highlighted in red and blue, respectively. None of the vehicles can serve the request on time if they can only accept small detours (e.g., 5 minutes of additional trip time for other passengers). Nevertheless, the passenger can arrive on time through a multi-hop trip: vehicle c_a carries the passenger from A to D and vehicle c_b carries the passenger from D to G.

Despite its advantages, multi-hop ride-sharing has only be applied to small networks for a limited number of vehicles and passengers [9–12] as they exhaustively enumerate all possibilities. Most of the existing multi-hop ride-sharing algorithms find multi-hop trips by searching for overlapping routes between passengers and vehic,les and enumerate all possible routes of vehicles and passengers, which leads to an exponential time complexity as even a slight detour generates a different route. Thus, previous methods are restricted to small road networks (at most 10000 nodes). Efficient solution for multi-hop ride-sharing in real-world scenarios for large city road networks remains challenging.

To fully exploit the benefits of multi-hop ride-sharing, we propose scalable real-time multi-hop dispatching algorithms for large scale real-world scenarios. We propose pruning strategies to reduce the search space using the time constraints of requests, thus achieving real-time responses. We propose two algorithms that cater for different scenarios: *Station-first* and *Vehicle-first.*

The Station-first algorithm first computes possible transfer points and then finds feasible vehicles. Its efficiency depends on the number of transfer points and is preferable when the transfer points are sparse. However, it might be less desired when more transfer points with a higher density are available. The Vehicle-first algorithm first prunes candidate vehicles that can serve a request (stage one) and then computes an optimal transfer point for each candidate trip (stage two). Based on the key observation that the time constraints of the committed requests limit the area (called the *reachable area*) that a vehicle can reach, we prune vehicles by computing whether or not their reachable areas cover the new request. We then solve the problem of finding an optimal transfer point by reducing it into a variation of a group nearest neighbor query problem [13]. The Vehicle-first algorithm avoids checking on all possible transfer points and achieves higher efficiency with dense transfer points.

Since in real applications the provision of real-time responses to requests may be more important than finding the optimal trip, e.g., when it rains and a passenger quickly needs to find a trip. We propose two strategies to accelerate the Vehicle-first algorithm. The first strategy reduces the number of paired vehicles returned in stage one. In the exact Vehicle-first algorithm, we represent reachable areas of vehicles through ellipses (denoted as *bounding ellipses*) and pair vehicles if their bounding ellipses overlap. As the actual reachable areas are often smaller than the bounding ellipses due to the road network constraints, we infer the actual reachable areas instead of using the ellipses, which leads to fewer overlapping vehicles and candidate trips. Recent studies [14, 15] show that deep learning has great potential in computing road network distances. Thus, we predict the actual reachable areas using deep learning. We achieve accurate predictions by integrating the bounding ellipses. The second approximation strategy is performed in the second stage of the Vehicle-first algorithm, i.e., computating the optimal transfer points. Instead of exhaustively checking all potential transfer points, we only check a few estimated transfer point that seems optimal, which substantially reduces the search time.

The contributions of this paper are summarized as follows:

- We propose scalable real-time multi-hop ride-sharing dispatching algorithms. To the best of our knowledge, this is the first work applicable to real-world ride-sharing scenarios with large sets of transfer points.
- We propose two algorithms to cater for different application scenarios. Both algorithms apply efficient and effective pruning to reduce the search space and achieve high efficiency.
- We propose approximation strategies to reduce response times by utilizing deep learning and efficient indices.
- We experimentally verify the benefits of multi-hop ride-sharing and demonstrate the efficacy and efficiency of the proposed algorithms over real-world datasets. Our algorithms are two to three orders of magnitude faster than the state-of-the-art. The efficiency can be improved by another order of magnitude if applying approximation strategies.

2 RELATED WORK

Studies of ride-sharing algorithms originate from the static dial-a-ride problem [16] and have developed towards large-scale and real-time application scenarios.

Single-hop ride-sharing. Most studies on ride-sharing have focused on the *single-hop dispatching* problem where every passenger is transported by only one vehicle [1–6, 17–19]. Single-hop dispatching algorithms typically work in two stages – pruning and selection. The pruning stage quickly filters out infeasible matches violating the constraints of requests. The selection stage processes the remaining vehicles to find an optimal match based on given optimization goals. Popular optimization goals include minimizing the total travel distance of vehicles [1, 5, 6, 17] and maximizing the system profit [4, 18, 19].

Multi-hop ride-sharing. Existing multi-hop algorithms mostly model the problem using *time-expanded graphs* (TEG) [9, 10, 12]. In a TEG, the TEG nodes have two keys: location id and time, representing that the vehicle can reach the location at the specified time. The TEG nodes record possible visits of vehicles with time information. These nodes are connected by TEG edges to denote possible connections such that possible routes can be tracked following the edges. A TEG edge that connects two TEG nodes of different locations is called a *transfer edge*, whereas a *waiting edge* connects two TEG nodes of the same location but different timestamps.

The first adoption of TEG in multi-hop ride-sharing is [12]. They reduce the problem of multi-objective optimization into a multi-objective shortest path problem and solve it using an evolutionary algorithm after building the TEGs. Several algorithms are proposed to accelerate the search of multi-hop matches on TEG graphs. Drews et al. [10] apply A* algorithm on the TEG graph to find the best multi-hop option. Masoud et al. [9] reduce the number of nodes in a TEG by only considering points within areas bounded by ellipses. They further develop dynamic programming algorithms to accelerate search on TEGs for the best routes [9]. Despite the reduced search spaces, their algorithms are still based on TEGs and thus are only able to handle small road networks and few transfer points (the maximum evaluated network has only 10000 nodes). Besides, they assume a vehicle can at most occupy one more request along a pre-defined path, while we study a real-world taxi ride-sharing scenario in which vehicles are roaming in the street and can serve multiple requests at a time.

Several other studies assign multiple trip requests jointly and focus on improving the defined optimization goals instead of the matching efficiency. Hou et al. [20] aim to optimize the matching ratio. They first enumerate the multi-hop options for all passengers and then assign requests to passengers ordering by the developed strategies, e.g., first assign requests requiring less detour. Coltin et al. [21] also enumerate all multi-hop options before dispatching, aiming to minimize the total travel distance of all vehicles and the transfer costs of requests. None of these works can apply to real-world scenarios due to the overhead of the exhaustive search.

Herbawi et al. [11] model the problem using time windows and apply a similar idea to our Station-first algorithm (Section 4), i.e., reducing the multi-hop dispatching problem to single-hop. However, they employ genetic algorithms that are only applicable to small networks (only 50 vehicles and 150 requests are experimentally evaluated), while we adopt a state-of-the-art single-hop dispatching algorithm that is more scalable and efficient [6].

Group nearest neighbor query. Given a set of query points and a set of data objects, a *group nearest neighbor* (GNN) query finds the nearest object with the minimum sum (or max) distance

to all query points. GNN and its variants have been studied in the literature [13, 22–24]. GNN algorithms can be categorized into two classes: *incremental network expansion* (INE) and *incremental Euclidean distance restriction* (IER). INE algorithms gradually expand the search space from the query points until the optimal object is found. Instead, the IER algorithms use the Euclidean GNN distances as lower bounds to guide the search towards the optimal network GNN object. The IER algorithm achieves faster response time in most cases except when the query points are dense [13, 22, 23].

3 PRELIMINARIES

3.1 Basic Concepts

We consider a multi-hop ride-sharing system on a road network which is represented as a directed graph $G = \langle V, E \rangle$, where V denotes a set of nodes and E represents a set of edges connecting these nodes. Each edge $e(v_i, v_j)$ is associated with a weight representing the travel cost from node v_i to v_j. We denote their travel distance as $d(v_i, v_j)$ and their travel time as $t(v_i, v_j)$. A location is denoted as a *transfer point* if transfer at the location is allowed.

Trip request. A *trip request* $r_i = \langle t, s, e, \tau_s, \tau_e, \eta \rangle$ has six elements: the issue time t; the source location s, the destination location e, the latest pickup time τ_s, the latest dropoff time τ_e and the number of passengers η. A set of trip requests is represented as $R = \{r_1, r_2, ..., r_n\}$. Similar to previous works [1, 5, 6], we use the maximum waiting time $r_i.w$ and detour ratio $r_i.\epsilon$ to define the latest arrival times: $\tau_s = t + w$, $\tau_e = t + w + t(s, e) \times (1 + \epsilon)$.

Vehicle. A *vehicle* $c_i = \langle l, S, u, v \rangle$ has four elements, the current location l, the trip schedule S, the vehicle capacity u, and the vehicle speed v. A set of vehicles is represented as $C = \{c_1, c_2, ..., c_n\}$.

Vehicle schedule. We denote the trip schedule of vehicle c_i as a sequence of locations $c_i.S = \langle p^0, p^1, p^2, ..., p^m \rangle$, where p^i is a node in the road network representing a *stop* that is a source, destination or transfer point of an assigned request. We distinguish stops locating at the same place considering their different time constraints and request information. A vehicle not committed to any request has only one stop in its trip schedule (current location p^0). We call the route between two consecutive stops a *segment*.

Following previous works [5, 6, 17], we maintain three arrays to record the trip schedule information: $arr[]$ records the earliest arrival time of each stop, $lat[]$ records the latest arrival time of each stop and $slk[]$ records the maximum allowed detour time before each stop to ensure the service constraints of all passengers.

$arr[]$ is calculated by accumulating the shortest travel time between stops, i.e., $arr[k] = arr[k-1] + t(p^{k-1}, p^k)$. $lat[]$ is the latest arrival time of the referred stop, restricted by the corresponding request. $lat[k] - arr[k]$ specifies the maximum allowed detour time before stop k respecting the latest arrival time of the stop k. A detour before p^k delays the arrival time of all following stops of p^k. $slk[k]$ records the maximum allowed detour time satisfying the arrival times of the stop k and that of all stops scheduled after k, i.e., $slk[k] = min(lat[i] - arr[i]), i = k, ..., m$. The maximum allowed travel time between a segment (p^{k-1}, p^k) is thus computed as $max(p^{k-1}, p^k) = arr[k] - arr[k-1] + slk[k]$.

After a stop p^k, the vehicle can only visit a restricted area to satisfy the maximum allowed travel time of the segment (p^k, p^{k+1}), i.e., $max(p^k, p^{k+1})$. We denote such an area as the *reachable area* after p^k, i.e., $reach(p^k)$. The slack time and maximum allowed travel time after the last stop p^m are infinite, and the reachable area after the last stop p^m covers the whole space.

Similar to previous studies [5, 6, 17, 25, 26], we keep the order of existing trip schedules and insert new stops to the exiting schedules. The *insertion position* k indicates that the new stop is inserted after the stop p^k. Note that insertions cannot before p^0 that represents the current location of the vehicle.

A trip schedule is *valid* if it satisfies the following conditions:

- Stop order. A vehicle needs to visit the source of a request before visiting the transfer point or visit the transfer point before visiting the destination of the request.
- Time constraint. The estimated arrival time of a stop representing the source or destination of a request must be no later than its latest arrival time.
- Capacity. The number of on-board passengers cannot exceed the capacity of the vehicle at any time.

3.2 Problem Definition

We focus on direct trips and trips with one transfer point (i.e., 2-hop trips), since previous studies [9, 10, 20, 27] have shown that allowing more than one transfer in a trip brings marginal benefits.

For a direct trip, a vehicle will be dispatched to pick up the request and deliver the passenger(s) to their destination directly. As for a multi-hop (i.e., 2-hop) trip, two vehicles will be dispatched to the request with a transfer point assigned. The first vehicle carries the request from the source to the transfer point where the request transfers to the second vehicle to continue the remaining trip until the destination is reached. We denote the trip from the source to the transfer point as the **first itinerary** and the trip from the transfer point to the destination as the **second itinerary**.

A **match** of a new request r_n, denoted by $r_n.m = \langle c_1, c_2, \phi, \Gamma \rangle$, consists of four elements: the vehicle carrying the first itinerary c_1, the vehicle carrying the second itinerary c_2, the transfer point ϕ, and the insertion positions $\Gamma = \{s, \phi_1, \phi_2, e\}$ including four values:

(1) $\Gamma(s)$: insertion position of the source to the first vehicle's schedule.
(2) $\Gamma(\phi_1)$: insertion position of the transfer point to the first vehicle's schedule.
(3) $\Gamma(\phi_2)$: insertion position of the transfer point to the second vehicle's schedule.
(4) $\Gamma(e)$: insertion position of the destination to the second vehicle's schedule.

We use ϕ_1 and ϕ_2 to distinguish the transfer points in the two vehicle schedules if necessary. For direct matches, c_2 equals to c_1, and the insertions related to transfers are null, i.e., $\Gamma(\phi_1)$, $\Gamma(\phi_2)$.

Example 3.1. In Figure 1, a multi-hop trip $r_n.m = \langle c_a, c_b, D, \Gamma = \{c_a.1, c_a.1, c_b.2, c_b.2\} \rangle$ of r_n indicates that r_n will be first carried by c_a and then transfer to c_b at the location D. Both source (A) and the transfer point (D) will be inserted after the first stop of c_a, updating the trip schedule from $A \rightarrow M$ to $A \rightarrow \mathbf{A} \rightarrow \mathbf{D} \rightarrow M$. Similarly, both the transfer point (D) and the destination (G) will be inserted after the second stop of the vehicle, updating the trip schedule from $B \rightarrow D \rightarrow H$ to $B \rightarrow D \rightarrow \mathbf{D} \rightarrow \mathbf{G} \rightarrow H$.

For a multi-hop trip, the assigned two vehicles may reach the transfer point at different times. If the first vehicle arrives earlier, the passengers have to stay at the transfer point to meet the second vehicle. If the second vehicle arrives earlier, the second vehicle needs to wait at the transfer point until the first vehicle comes.

A feasible multi-hop match. A feasible multi-hop match should satisfy the time constraints of the new request and all existing requests. We consider the following service constraints:

- r_n must be picked up and dropped off on time, i.e., $arr[\Gamma(s)] + t(p^{\Gamma(s)}, r_n.s) \leq r_n.\tau_s$; $arr[\Gamma(e)] + det(\phi_2) + t(p^{\Gamma(e)}, r_n.e) \leq r_n.\tau_e$, where $det(x)$ denotes the additional trip time to visit the location x. $det(\phi_2)$ includes waiting time of the second vehicle at ϕ if necessary.
- The detour time of the first vehicle cannot exceed its maximum allowed detour time to ensure the latest arrival times of all committed requests, i.e., $det(r_n.s) \leq slk[\Gamma(s)+1]$; $det(r_n.s) + det(\phi_1) < slk[\Gamma(\phi_1)+1]$.
- The detour time of the second vehicle cannot exceed its maximum allowed detour time, i.e., $det(\phi_2) \leq slk[\Gamma(\phi_2)+1]$; $det(\phi_2) + det(r_n.e) \leq slk[\Gamma(e)+1]$.

Matching Objective. Although our proposed algorithms are applicable to other optimization goals, we simplify the discussion by studying a popular optimization objective – minimizing the travel distance of vehicles [1, 5, 6, 17]. Assume that the overall scheduled travel distance of all vehicles was T before dispatching requests, and the distance becomes T' after dispatching all requests $R = \{r_1, r_2, ..., r_n\}$, the goal is to minimize the additional distance $T' - T$ to serve all requests.

As the optimization problem is NP-hard [1, 17] we apply a popular strategy [1, 5, 6, 17] that sequentially matches passengers ordered by their issue times. For every new request, we dispatch an optimal trip that minimizes the additional travel distance $T' - T$ to serve it. We compute the optimal direct trip using the state-of-the-art single-hop dispatching algorithm [6] and the optimal multi-hop trip using the proposed multi-hop dispatching algorithm. An optimal one among them is assigned to the request.

We keep dispatches unchanged once allocated and assume vehicles to follow the scheduled routes. We leave the discussion of handling incidents such as cancellation of requests for future work.

4 STATION-FIRST ALGORITHM

Next, we present our Station-first algorithm that first identifies potential transfer points and then searches for possible vehicles by adopting the state-of-the-art single-hop algorithm GeoPrune [6].

4.1 GeoPrune

When a new request r_n arrives, GeoPrune computes a *waiting circle* circle(r_n.s) and a *detour ellipse* ellipse($r_n.s$, $r_n.e$) to assist searching for possible vehicles. The waiting circle bounds the locations of potential vehicles. The detour ellipse indicates the area that the request may detour to ensure the latest arrival time at the destination. The circles and ellipses are represented by their minimum bounding rectangles (MBRs) for quicker retrieval and updates.

(1) **circle (p_1)**. The waiting circle of p_1 is centered at p_1 with radius equal to (max_speed × ($lat[p_1]$ - curr_time)).

(2) **ellipse** (p_1, p_2). The detour ellipse between p_1 and p_2 considers p_1 and p_2 as two focal points and the major length being (max_speed × $max(p_1, p_2)$).

max_speed is the maximum vehicle speed, $lat[p_1]$ is the latest arrival time at p_1, curr_time is the current system time, and $max(p_1, p_2)$ is the maximum allowed travel time between p_1 and p_2.

During the matching process, GeoPrune records the reachable area between every two consecutive stops (a segment) using a detour ellipse. Indexing these detour ellipses enables a quick assessment of whether a location is reachable by a vehicle. GeoPrune finds the following vehicles as candidates to serve r_n (pruning stage):

(1) a vehicle with its last stop covered by circle(r_n.s);
(2) a vehicle with at least one detour ellipse of its existing schedules covering the source $r_n.s$ and at least one detour ellipse of its existing schedule covering the destination $r_n.e$;
(3) a vehicle with at least one detour ellipse of its existing schedule covering the source $r_n.s$ and its last stop covered by the request detour ellipse.

After obtaining the vehicle candidates, GeoPrune checks the insertion feasibility to every vehicle candidate and selects the optimal match to return to the passenger (selection stage).

Example 4.1. Assume all nodes in the road network in Figure 1 are transfer points. The schedule of c_a is $A - M$ and the reachable area between $A - M$ is bounded by the red ellipse. The schedule of c_b is $B - D - H$, and the reachable areas of $B - D$ and $D - H$ are bounded by two blue ellipses, respectively. When a new request arrives, GeoPrune constructs its waiting circle and detour ellipse (with dashed green boundaries). Assume the last stops of both vehicles are outside of the waiting circle, then no vehicle satisfies the condition (1) listed above. As neither vehicle has the detour ellipses covering both the source and the destination, no vehicle satisfies the condition (2). The request detour ellipse covers the last stop of neither vehicle, so no vehicle satisfies the condition (3). Thus, the request cannot be served by a direct trip.

4.2 Multi-hop Station-First Algorithm

We next detail our multi-hop Station-first algorithm. The basic idea of the algorithm is to split a request trip by a transfer point and then apply the GeoPrune algorithm on the two generated itineraries.

Stage 1: Identify possible transfer points. We only examine transfer points within the detour ellipse, i.e., ellipse($r_n.s$, $r_n.e$). Visiting any points outside of the request detour ellipse will violate the latest arrival time of the request [6, 9] and thus points outside of the ellipse cannot be a transfer point. We build an R-tree on all possible transfer points on the road network (T_{tsf}) and run a range query using ellipse($r_n.s$, $r_n.e$) to retrieve the possible transfer points.

Stage 2: Check each possible transfer point. We then check the feasibility of every possible transfer point. A transfer point ϕ splits the request trip into two itineraries $(r_n.s, \phi)$ and $(\phi, r_n.e)$. We derive their time constraints so as to apply the single-hop algorithm.

For the source $r_n.s$, the earliest arrival time ($ear()$) is the request issue time, and the latest arrival time ($lat()$) is the latest pickup time, i.e., $ear(r_n.s) = r_n.t$, $lat(r_n.s) = r_n.\tau_s$. For the destination $r_n.e$, the earliest arrival time is the request issue time plus the shortest trip time of the request, i.e., $ear(r_n.e) = r_n.t + t(r_n.s, r_n.e)$. The latest arrival time is the latest drop-off time, i.e., $lat(r_n.e) = r_n.\tau_e$.

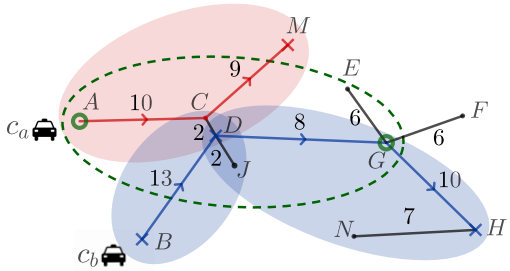


Figure 2: Detour ellipses of Figure 1.

The time constraints of the transfer point ϕ depend on its location and are different in the two itineraries. The earliest arrival time of ϕ in the first itinerary (ϕ_1) is the issue time of the request $r_n.t$ plus the direct trip time from $r_n.s$ to ϕ, i.e., $ear(\phi_1) = r_n.t + t(r_n.s, \phi)$. The system must reserve a time longer than the direct trip from ϕ to $r_n.e$ to ensure the latest arrival time of the request. Thus, the latest arrival time at ϕ is $lat(\phi_1) = r_n.\tau_e - t(\phi, r_n.e)$.

We apply GeoPrune to compute all feasible matches to serve the first itinerary before setting time constraints for the second itinerary. The selection stage needs to enumerate all insertion positions of source and destination for every vehicle candidate and find all feasible insertion positions. The first itinerary may find multiple matches with different estimated arrival times at the transfer point ($est(\phi_1)$) and thus define different second itineraries.

The transfer point is regarded as a source in the second itinerary (ϕ_2). Its earliest arrival time is its estimated arrival time in the first itinerary, i.e., $arr(\phi_2) = est(\phi_1)$. Its latest arrival time is the same as that of the first itinerary, i.e., $lat(\phi_2) = lat(\phi_1) = r_n.\tau_e - t(\phi, r_n.e)$.

We again apply GeoPrune on each generated second itinerary to compute all feasible matches. The combination of a feasible match of the first itinerary and that of the corresponding second itinerary forms a feasible multi-hop match. Among all feasible multi-hop and direct matches, we select and return the optimal one to the request.

Algorithm complexity. Assume there are $|P|$ transfer points, $|C|$ vehicles and the maximum number of stops of the vehicle schedule is $|S|$. For each transfer point, GeoPrune is applied once on the first itinerary. The maximum number of feasible matches of the first itinerary is $|C||S|^2$ as there are $|C|$ vehicles and each vehicle has $|S|$ positions to insert the source and $|S|$ positions to insert the destination. Each feasible match of the first itinerary defines a second itinerary and each generated second itinerary conducts GeoPrune once (at most $|C||S|^2$). Every second itinerary may also has at most $|C||S|^2$ insertion positions. Thus, the GeoPrune will be conducted at most ($|C||S|^2 + 1$) times for each transfer point and at most $|C|^2|S|^4$ insertion positions will be checked with $O(1)$ checking time. Combining with the GeoPrune complexity $O(\sqrt{|C||S|} + |C||S| \log(|C||S|))$, the overall complexity of the Station-first algorithm is $O(|P|(|C|^2|S|^4 + |C|^2|S|^3 \log(|C||S|)))$.

5 VEHICLE-FIRST ALGORITHM

The Station-first algorithm examines all possible transfer points, which may become expensive when there are many such points. Next, we propose an algorithm called Vehicle-first that examines fewer transfer points while preserving exact solutions.

The intuition is that two vehicles can be paired up in a trip only if their reachable areas (ellipses) overlap and the transfer points must be within the detour ellipses of both vehicles and the request. By using efficient data structures, we can quickly locate overlapping ellipses and identify possible vehicle pairs and insertion positions.

A vehicle pair and the specified insertion positions form a multi-hop trip candidate. Our further analysis shows that the optimal transfer point of each trip candidate only depends on three or four stops of vehicles, which enables us to reduce the problem into a variation of group nearest neighbor query (GNN) in road networks. Repeatedly running GNN queries on each trip candidate lacks efficiency when there are many candidates vehicle pairs. We hence propose a novel algorithm to accelerate this process. The algorithm can be easily extended to other goals by customizing the algorithm of finding the optimal transfer points.

5.1 Stage 1: Find Possible Insertion Positions

We first explain how to identify possible vehicle pairs and the insertion positions of source, destination and the transfer point. We first compute *circle($r_n.s$)* and *ellipse($r_n.s$, $r_n.e$)* for a given request r_n and then run a two-phase refinement process to find the possible trip candidates. The first phase locates the possible insertions of the source $r_n.s$ and destination $r_n.e$ to a vehicle schedule. The second phase detects the overlap areas between vehicles and identify possible insertions of the transfer point.

Phase 1: Identify possible insertions of source and destination. There are two types of insertion positions to add a new stop to the schedule of a vehicle: *insert-between* and *insert-after*. An insert-between position indicates an insertion between two existing stops of the vehicle schedule while an insert-after position indicates appending the new stop after the last stop.

The same as GeoPrune [6], we construct two R-trees to index the detour ellipses and the last stops of vehicles respectively: T_{seg} and T_{end}. We run four queries to identify possible insertions of the source and destination (and thus the possible vehicles).

(1) insert-between for source $r_n.s$: querying segments with the detour ellipse covering $r_n.s$, $T_{seg}.pointQuery(r_n.s)$.
(2) insert-after for source $r_n.s$: querying vehicles with the last stop covered by circle($r_n.s$), $T_{end}.rangeQuery(circle(r_n.s))$.
(3) insert-between for destination $r_n.e$: querying segments with the detour ellipse covering $r_n.e$, $T_{seg}.pointQuery(r_n.e)$.
(4) insert-after for destination $r_n.e$: querying vehicles with the last stop covered by circle($r_n.e$), $T_{end}.rangeQuery(circle(r_n.e))$.

We discard an insertion position if it violates the time constraints of any stop. Assume that the previous stop in the new schedule is p^k and the new stop is p, if the estimated arrival time of p (computed by summing up the estimated arrival time of p^k and the travel time from p^k to p) is later than its allowed latest arrival time, we discard the insertion position. Besides, if the insertion position refers to a segment (p^k, p^{k+1}), we discard the insertion position if the caused detour time is larger than the slack time of the segment. The range query $T_{end}.range(circle(r_n.e))$ may return many vehicles due to the possibly large querying circle. However, most of the vehicles may have late arrival times at their last stops and the last stops may be far from the destination. Checking the time constraints helps to safely discard these infeasible vehicles.

Phase 2: Identify overlap reachable areas. After running the four queries, vehicles that can be scheduled to visit the source or the destination of the request are obtained. In Phase 2, we pair up

such vehicles by detecting the overlap reachable areas of them by the constraints: 1) The transfer point must be within the detour ellipse of the request. 2) In the first vehicle's schedule, the insertion position of the transfer point must be **no earlier than** that of the source. 3) In the second vehicle's schedule, the insertion position of the transfer point must be **no later than** that of the destination.

For each pair of source insertion and destination insertion belonging to two different vehicles, we check if there is a reachable area *after* the source insertion overlaps with a reachable area *before* the destination insertion. If we detect an overlap within the request ellipse, we create a multi-hop trip candidate with the two vehicles and insertion positions specified. The possible transfer points must be within the overlapped reachable areas and the request ellipse. We denote such an area as the *transfer window* of the trip candidate.

Example 5.1. In Figure 2, we first search for vehicles that can reach source A and destination G, respectively. For source A, only the first ellipse of c_a (the red ellipse) covers A. Assuming the waiting circle of $r_n.s$ covers the last stop of neither vehicle, A can only be inserted after the first stop in c_a's schedule. As for destination G, the second ellipse of c_b (the blue ellipse on the right) covers G. Assuming the waiting circle of G covers both last stops of c_a and c_b, G has three possible insertion positions: after the second stop of c_b, after the last (third) stop of c_b, and after the last (second) stop of c_a. Next, we check the time constraints of these insertion positions. We assume that inserting G to the last stop of c_b violates the latest arrival time of G and discard this insertion possibility. We then pair up the possible insertion positions of A and G. The insertion pair $(\Gamma(s), \Gamma(e)) = (c_a.1, c_a.2)$ is infeasible because the source and destination must be served by two different vehicles. Thus, only one insertion pair is remained: $(\Gamma(s), \Gamma(e)) = (c_a.1, c_b.2)$.

We then check the transfer possibility for the remaining insertion pair $(\Gamma(s), \Gamma(e)) = (c_a.1, c_b.2)$. We check if there is any reachable area after the first stop of c_a overlap with that before the second stop of c_b. Since the first ellipse c_a overlaps with the first and second ellipses of c_b, the possible insertions of the transfer point are: 1) after the first stop of c_a and after the first stop of c_b, 2) after the first stop of c_a and after the second stop of c_b, yielding two multi-hop trip candidates: 1) $\langle c_a, c_b, \phi, \Gamma = (c_a.1, c_a.1, c_b.1, c_b.2)\rangle$; 2) $\langle c_a, c_b, \phi, \Gamma = (c_a.1, c_a.1, c_b.2, c_b.2)\rangle$. Their optimal transfer points are not decided yet and bounded by the overlap areas.

5.2 Stage 2: Find the Optimal Transfer Point

The remaining challenge is how to quickly find an optimal match with optimal transfer point. A naive solution of finding the optimal transfer point of a trip candidate is to check all transfer points within the transfer window. Such a solution is inefficient when many transfer points are possible. Next, we reduce the problem into a variation of the GNN query to enable a more efficient solution.

5.2.1 Problem Reduction. Given a multi-hop candidate, the *additional distance* equals to the travel distance of the new schedules of the two vehicles *minus* the travel distance of their existing schedules. As the travel distance of the existing schedules is a constant, the problem of minimizing the additional distance is reduced to minimizing the travel distance of the new schedule.

If we append both source and transfer point to the first vehicle's trip schedule, the transfer point will be the last stop and only the source stop will be connected to the transfer stop in the new schedule. If we insert both source and the transfer point in the middle of the first vehicle's schedule, there will be two stops connecting the transfer point, one at the front and one afterward. On the other hand, in the schedule of the second vehicle, the destination stop must be placed after the transfer stop. Hence, the transfer stop cannot be the last stop and there must be two stops connecting the transfer point in the second vehicle's schedule.

We denote the stops connecting the transfer point in the new schedule as **GNN stops** and the sum of distances from a point to the GNN stops as the **GNN distance**. The additional distance of a multi-hop match can be computed as the sum of two parts: a constant part that equals to the travel distance connecting non-GNN stops in the new schedules *minus* the travel distance of the existing trip schedules, and a variable part representing the GNN distance. Finding the optimal transfer point is thus reduced to finding a transfer point with minimum sum of distances to three or four fixed (GNN) stops while satisfying the time constrains, which is a variation of a group nearest neighbor (GNN) in a road network.

Example 5.2. Consider a multi-hop option $\langle c_a, c_b, \phi, \Gamma = (c_a.1, c_a.1, c_b.2, c_b.2)\rangle$. The schedule of c_a changes from $A - M$ to $A - A - \phi - M$. The travel distance of the existing schedule $A - M$ is a constant $d(A, M)$. The travel distance of the new schedule equals to $d(A, \phi) + d(\phi, M)$ ($d(A, A) = 0$), which is changing with the choice of ϕ. Similarly, the schedule of c_b changes from $B - D - H$ to $B - D - \phi - G - H$. The travel distance of the existing schedule is a constant $d(B, D) + d(D, H)$. The travel distance of the new schedule is $d(B, D) + d(D, \phi) + d(\phi, G), d(G, H)$. $d(B, D)$ and $d(G, H)$ are fixed while $d(D, \phi) d(\phi, G)$ vary based on ϕ. The problem of minimizing the additional distance is reduced to minimizing $d(A, \phi) + d(\phi, M) + d(D, \phi) + d(\phi, G)$, which is further reduced to finding a feasible transfer point to minimize the sum distance to A, M, D, G.

5.2.2 Collaborative IER. Applying an existing GNN algorithm to each multi-hop trip lacks efficiency when many multi-hop candidates remains. Observing that many multi-hop trips may share overlap transfer windows that will be searched multiple times and cause redundant computation, we propose to collaboratively process all multi-hop candidates while only explore the space once.

IER. We first describe an existing GNN algorithm *Incremental Euclidean Restriction* (IER) [13]. IER traverses the R-tree indexing all transfer points from top to bottom. Given an R-tree node *Rnode*, the GNN distance of any point indexed under *Rnode* must be larger than the sum Euclidean distance from the GNN query points (GNN stops in our problem) to the minimum bounding box of *Rnode*. To exploit this property, the algorithm maintains a priority queue to sort R-tree nodes by their sum Euclidean distance to the GNN query points. Initializing the queue as the root of T_{tsf}, IER iteratively extracts minimum elements from the queue and inserts its children nodes to the queue. Hence, points with smaller Euclidean GNN distances will be visited first. If the extracted element refers to a transfer point, we check its feasibility (Section 3.2) and update the optimal GNN distance $best\_dist$. If the key of the next element in the queue is larger than $best\_dist$, IER terminates as the sum network distance of any unchecked nodes must be larger than the current best result.

Collaborative IER. The basic idea is that when reaching an R-tree node, instead of computing the GNN lower bound of a single

multi-hop candidate, we consider the lower bounds of **all** multi-hop candidates. The algorithm is guided to first visit the transfer points with smaller lower bounds respecting all multi-hop candidates, which enables earlier termination and reduced search space.

Every R-tree node maintains three additional variables during the traversal: active multi-hop candidates (*active_cand*), lower bound of these active candidates (*active_LB*), and the minimum lower bound of these active candidates(*min_LB*). *active_cand* stores all multi-hop candidates with their transfer window intersecting the MBR of the R-tree node, while all other multi-hop trips are infeasible to transfer within the indexed area. For each candidate in *active_cand*, the lower bound of minimum additional distance considering any transfer points within the indexed area is recorded in *active_LB*. Note that the recorded lower bound of a multi-hop candidate is the lower bound of the minimum additional distance that considers both the GNN distance and the constant part. The minimum *active_LB* of all active multi-hop candidates is indicated by *min_LB*.

The same as IER, we maintain a priority queue that sorts R-tree nodes by their minimum lower bound *min_LB*. We initialize the queue with the R-tree root and iteratively dequeue top elements. After each dequeue, we process the children nodes of the dequeued node. We remove those multi-hop candidates that are no longer active in the children nodes if their transfer windows become non-overlapping with the indexed. We also update the lower bounds of all active candidates for each child node. We insert a child node to the queue if its active multi-hop is not empty and set the key as the updated minimum lower bound. If the extracted node is a leaf node, we check the feasibility of the corresponding transfer point considering all its active multi-hop candidates. We update the optimal additional distance *best_add_dist* if a smaller distance is achieved. When the key of the top element in the queue exceeds the recorded optimal distance *best_add_dist*, we terminate the search and return the optimal multi-hop match and optimal transfer point.

Algorithm complexity. Querying the possible insertion positions of source and destination takes $O(\sqrt{|S||C|} + |S||C|)$ time. The source (or destination) has $|C||S|$ possible insertion positions, each of which has at most $|S|$ positions to insert the transfer position. The overall number of trip candidates is thus at most $|C|^2|S|^4$. In the worst case, the algorithm visits all nodes during the R-tree traversal ($O(|V|+\log|V|)$) and each node processes all routes ($O(|V||C|^2|S|^4)$). The overall complexity is therefore $O(\log|V| + |V||C|^2|S|^4)$.

5.3 Performance Enhancement Through Deep Learning and Approximation

The above algorithms guarantee the finding of the optimal multi-hop trip. We propose two approximation strategies to accelerate the two stages of the Vehicle-first algorithm while achieving near-optimal matches. The first strategy employs deep learning to shrink the representation of reachable areas such that fewer vehicles will overlap and be paired up. The second strategy approximates the optimal transfer point such that fewer points need to be checked.

5.3.1 Learning the Reachable Area. The exact algorithms use ellipses to bound the reachable areas and guarantee the pruning correctness. However, it may bring extra computation as the actual reachable areas considering the network distance are usually smaller than the ellipses. If we use actual reachable areas instead of the ellipses in Vehicle-first algorithm, fewer trip candidates may survive the pruning, thus improving the query efficiency. Computing the actual reachable areas online is costly due to the expensive graph traversal. We predict the reachable areas using deep learning considering the high accuracy and low prediction cost [14, 15].

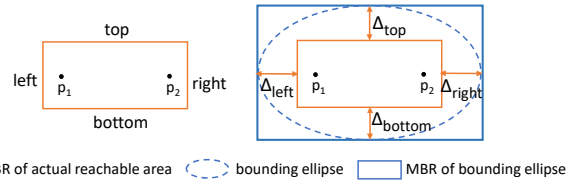


Figure 3: Reachable area prediction (left: Boundary Prediction; right: Gap & GapCustom Prediction).

The model is a multi-layer feed-forward network with 12 hidden layers. The architecture follows encoder-decoder architecture style where number of neurons in each layer is (64, 64, 128, 128, 256, 256, 256, 256, 128, 128, 64, 64). We use ReLU as the activation function for the hidden layers and linear for the output layer.

Given a road network, a reachable area is determined by three factors: source, destination, and time budget. Each location has two coordinates (longitude and latitude) and thus an input to our neural network is comprised of five elements: *source_lon*, *source_lat*, *dest_lon*, *dest_lat*, *time_budget*. For fast retrieval and updates, we still represent reachable areas as rectangles. The prediction goal is thus to obtain the four boundaries of a rectangle, i.e., $left$, $bottom$, $right$, top. We consider the following three prediction strategies:

Boundary prediction. The strategy predicts the four boundaries of the actual reachable rectangle directly, as shown in the left of Figure 3. The training uses the Mean Square Error (MSE) as the loss function to minimize the differences between the predicted boundaries and the actual boundaries.

Gap prediction. This strategy uses the bounding information provided by the MBRs of ellipses, which is inspired by the key observation that the actual reachable area is always a sub-area of the bounding ellipse (rectangle). Instead of predicting the boundaries directly, for each boundary, we predict the gap between the MBR of the bounding ellipse and the MBR of the actual reachable area: Δ_{left}, Δ_{bottom}, Δ_{right} and Δ_{top} (right of Figure 3). We use the MSE as the loss function to minimize the difference between the predicted gaps and the actual gaps when training. Let the boundaries of the bounding ellipse be *bound_left*, *bound_right*, *bound_top*, and *bound_bottom*, the predicted rectangle can be inferred as:

(1) $pred\_left = bound\_left + \Delta_{left}$;
(2) $pred\_right = bound\_right - \Delta_{right}$;
(3) $pred\_bottom = bound\_bottom + \Delta_{bottom}$;
(4) $pred\_top = bound\_top - \Delta_{top}$.

GapCustom prediction. If the predicted gap is larger than the actual gap, the predicted area will be smaller than the actual reachable area. Thus, some feasible vehicle pairs (with overlapped actual reachable areas) may be missed if the predicted areas no longer overlap. To avoid such false negative pairs, we propose the GapCustom prediction that applies a customized loss function to the Gap prediction and penalizes predictions larger than the actual gaps: $\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} \varphi(y\_true - y\_pred))^2$.

where φ is the penalty factor: $\varphi = 1$ if $y\_pred \leq y\_true$ whereas $\varphi > 1$ if $y\_pred > y\_true$. In our experiments, we choose the penalty factor $\phi = 10$ for larger prediction.

The predicted boundaries can not produce a closed rectangle if $pred\_left > pred\_right$ or $pred\_bottom > pred\_top$. For such cases, we still represent the reachable area using the bounding ellipse. Besides, as the applied gap value must be non-negative, we treat a predicted negative gap as zero in the Gap prediction and GapCustom prediction, indicating no shrink on the boundary.

5.3.2 Quickly locating the optimal transfer point. The Vehicle-first algorithm checks the feasibility of multiple transfer points before returning an optimal feasible one. In an extreme case, no transfer point is feasible and the algorithm needs to check all transfer points within the transfer window. To accelerate the process, we propose an approximation strategy to check only a few transfer points for each vehicle pair. The basic idea is to estimate the optimal transfer point in the road network with the optimal Euclidean GNN transfer points. Specifically, for each candidate trip, we only check k transfer points ($k = 5$ in our experiments) with the top-k minimum sum Euclidean distance to the GNN stops, which can be quickly obtained by using data structures such as R-trees [13, 23].

6 EXPERIMENTS

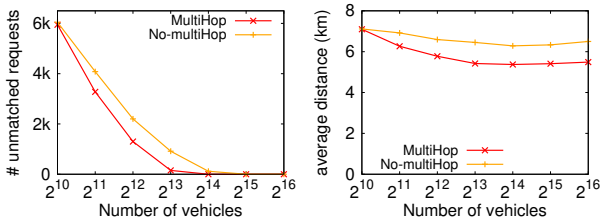
In this section, we study the empirical performance of the proposed algorithms. We run all experiments on Linux OS with 2.7 GHz CPU and 32 GB memory. We train the deep learning models using tensorflow in Python and implement all other algorithms in C++. We first investigate the benefits of allowing transfers in ride-sharing under different parameter settings and then compare the efficiency and effectiveness of the proposed algorithms.

6.1 Experimental Setup

Dataset. We show the experiments on a real-world road network dataset extracted from OpenStreetMap: Chengdu (CD) with 254,423 nodes and 467,773 edges. We apply a public real-world dataset of trip requests [28] on this dataset. We observe similar algorithm performance on another real-world road network dataset *New York City* (166,296 nodes and 405,460 edges) with real-world trip requests [29], and omit the results due to the space limit. We transform the coordinates into Universal Transverse Mercator (UTM) for pruning based on Euclidean distance. Similar to previous studies [1, 6, 17], the locations are mapped to their nearest road network nodes, and the number of passengers is assumed to be one per request. We iteratively sample the transfer points by applying the k-medoids clustering on the network nodes, i.e., the smaller sets of transfer points are generated by clustering large sets of transfer points.

We generate the training and testing datasets by sampling the source, destination, and time budget and normalize them before training the model. We generate 50,000 samples and randomly select 40,000 of them as the training dataset and the rest as the test dataset. We train our model using Tensorflow with the maximum number of training epochs specified as 5000. The learning rate and the batch size are selected as 0.001 and 100, respectively. We split 20% data from the training dataset as the evaluation dataset, which is used to evaluate the model at the end of each training epoch. We monitor the loss value on the validation dataset and terminate the training if the validation loss stays unimproved for more than 50 epochs.

Matching strategies. We consider both single-hop and multi-hop settings. *No-multiHop:* a ride-sharing system where no transfer is allowed during the matching. *MultiHop:* a ride-sharing system where both single-hop and two-hop trips may be returned.



(a) # unmatched requests. **(b) Average trip distance.**

Figure 4: Effect of the number of vehicles.

Comparing algorithms. We compare the following algorithms:
SF. the Station-first algorithm described in Section 4.
VF. the Vehicle-first algorithm described in Section 5.
SF-pred. SF plus reachable area prediction (Section 5.3.1).
VF-pred. VF plus reachable area prediction (Section 5.3.1).
VF-apxgnn. VF plus GNN approximation (Section 5.3.2).
VF-apxgnn-pred. VF plus both reachable area prediction and GNN approximation.

Metrics. We measure the following metrics: *# unmatched requests* – number of requests that cannot find feasible matches; *average trip distance* – average trip distance per request; *matching time* – average time required to respond a request.

Default setting. By default, we run experiments on a scenario when the vehicles are insufficient to serve all requests directly. We simulate on 4,096 vehicles with 1,000 transfer points and set the detour ratio and maximum waiting time of requests as 0.4 and $4min$.

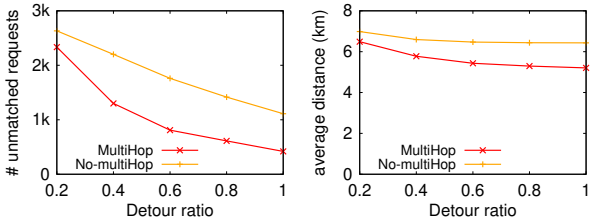
The state-of-the-art multi-hop matching algorithm [9] requires more than one week to match requests under the default setting. To compare the performance, we extract a small area from the central network of Chengdu (910 nodes and 1610 edges). We randomly generate 200 vehicles and 100 requests while considering all nodes as possible transfer points. Due to the hardness of enumerating all possible paths, [9] only computes k' shortest paths between two locations to construct the TEGs. Our experiments show that when $k' = 50$, [9] costs more than 42s while our algorithms can respond in 0.04s. Besides, the resulted total travel distance of [9] is almost twice as long as ours because of the k shortest path simplification.

6.2 Benefits of Multi-hop Trips

We first study the benefits of multi-hop trips with the effect of number of vehicles, detour ratio, and number of transfer points.

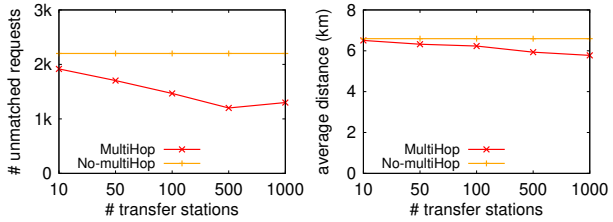
Effect of the number of vehicles. Figure 4 shows the effect of allowing multi-hop trips as the number of vehicles varies. Overall, enabling multi-hop creates more trip matches and reduces the average trip distance, which confirms that multi-hop is an effective strategy to improve the system performance. Interestingly, the matching quality diminishes for multi-hop with only a few vehicles (less than 2^{10}). A possible reason is that a small number of vehicles can be easily fully occupied even by direct trips, and the multi-hop trips lead to extra visits to transfer points.

Effect of the detour ratio. Figure 5 illustrates the effect of multi-hop with different detour willingness of passengers. The matching quality of multi-hop ride-sharing is observed to outperform that of single-hop ride-sharing in all cases. The advantage of multi-hop becomes more noticeable with higher detour ratios



(a) # unmatched requests. (b) Average trip distance.

Figure 5: Effect of the detour ratio.



(a) # unmatched requests. (b) Average trip distance.

Figure 6: Effect of the number of transfer points.

Prediction	Boundary	Gap	GapCustom
IoT	94.62%	93.25%	97.68%

Table 1: Prediction quality.

since passengers are more likely to share routes if they relax their time constraints. A reasonably high detour ratio may be observed in real-world applications for long-distance travelers who seek to save the travel cost by accepting longer arrival times.

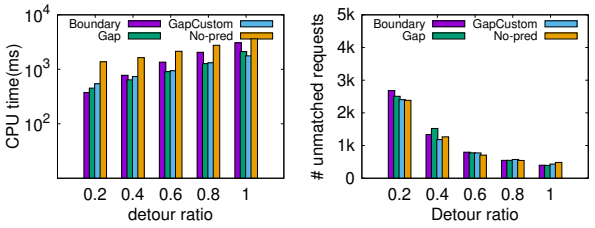
Effect of the number of transfer points. Figure 6 shows the system performance when varying the number of transfer points. Overall, providing more transfer points matches more requests and reduces the trip distance, despite slight fluctuations in the number of unmatched requests impacted by the greedy dispatching strategy.

6.3 Prediction Quality

We next investigate the prediction quality of the models proposed in Section 5.3.1. We use the metric *Intersection over True* (*IoT*) that is calculated by dividing the intersection between the actual area and the predicted area by the actual area. *IoT* indicates the fraction of the actual reachable area that is correctly predicted. A higher *IoT* implies more correctly predicted areas.

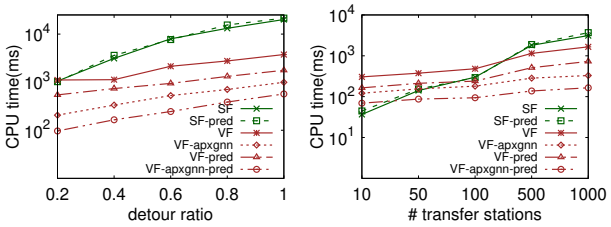
As shown in Table 1,all prediction strategies can successfully predict more than 90% of the actual reachable areas (as indicated by *IoT*), which confirms the effectiveness of applying deep learning to estimate the reachable areas. The GapCustom prediction yields the highest IoT because the customized loss function produces larger predicted areas within the bounding ellipses, while the Boundary prediction may predict many areas outside of the actual reachable areas and the Gap prediction may cause excessive shrinkage.

Figure 7 compares the performance of different prediction strategies on the ride-sharing dispatching process. In No-pred, the reachable areas are still represented as ellipses. All prediction strategies achieves comparable travel distance and we omit the results due to the space limit. All prediction strategies are observed to reduce the matching time by more than 50% while achieving comparable matching quality. GapCustom prediction yields the best balance between the matching time and the matching quality. It applies



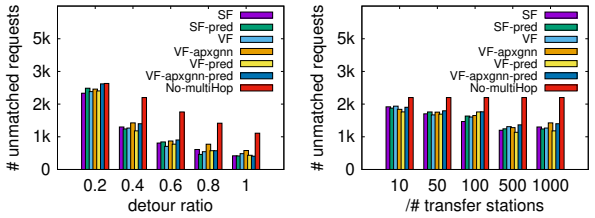
(a) Matching time (ms). (b) # unmatched requests.

Figure 7: Matching quality of prediction strategies.



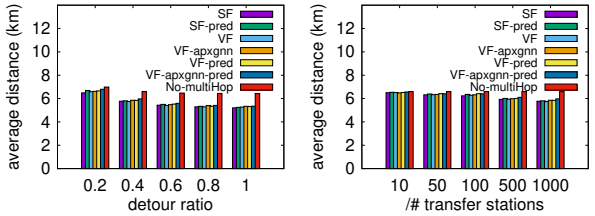
(a) Effect of the detour ratio. (b) Effect of the # transfer points.

Figure 8: Matching time.



(a) Effect of the detour ratio. (b) Effect of the # transfer points.

Figure 9: # unmatched requests.



(a) Effect of the detour ratio. (b) Effect of the # transfer points.

Figure 10: Average trip distance.

the customized loss function to improve the matching ratio of Gap prediction while requiring a slightly longer matching time. Gap prediction offers faster matching time than GapCustom prediction in almost all cases but results in fewer matched requests and longer travel distance. The Boundary prediction needs longer matching time in almost all cases except when the number of transfer points is 10, in which case the obtained number of matched requests is substantially smaller than other strategies.

6.4 Algorithm Performance

Figure 8 illustrates the performance of proposed algorithms. Figure 8a shows the effect of the detour ratio. All algorithms require longer time when increasing the detour ratio as more trip candidates need to be examined. The Station-first algorithm is more sensitive to the detour ratio. The reason is that its complexity largely depends

on the number of examining transfer points. A larger detour ratio results in a larger reachable area that covers more transfer points.

Figure 8b shows the effect of the number of transfer points on the matching time. The Station-first strategy is faster than the Vehicle-first strategy when the number of transfer points is small. However, its matching time becomes remarkably more expensive when the number of transfer points is increased, while the Vehicle-first strategy is barely affected. Therefore, the Station-first algorithm is shown to perform better for scenarios with only a few transfer points. When more transfer points are desired, the Vehicle-first algorithm becomes more efficient and favorable.

We evaluate the effectiveness of the reachable area prediction on both Station-first and Vehicle-first algorithms. Interestingly, the reachable area prediction largely reduces the matching time of the Vehicle-first algorithm but is unable to accelerate the Station-first algorithm. The reason is that large parts of the checking candidates are generated from the waiting circle of the second itinerary in the Station-first algorithm. Shrinking the reachable area cannot help to reduce the number of candidates.

As shown in Figure 8, both approximation strategies accelerate the Vehicle-first algorithm. The largest improvement is achieved when applying the two strategies together (one more order of magnitude faster compared to the exact Vehicle-first algorithm). Surprisingly, predicting the reachable areas improves the number of matched requests for both Vehicle-first and Station-first algorithms slightly (as shown in Figure 9). The reason might be that the smaller reachable areas reduce the matching possibility of requests visiting remote areas, leading to more vehicles moving around the central areas where more trips are requested and served. Approximating the transfer point saves more matching time than the reachable area prediction. It largely reduces the cost of finding optimal transfer points that requires expensive shortest path computations. Compared to only checking one optimal transfer point in Euclidean space, checking top-k points reduces missing trips and improves approximation quality. Applying approximation strategies achieves comparable travel distance of both algorithms (Figure 10).

7 CONCLUSION

We studied a real-time and scalable multi-hop ride-sharing system that allows transfers between vehicles. Our experiments show that offering multi-hop trips can increase service request matching ratio by up to 10% while reducing the average trip distance by 12%. Such an improvement enables more than ten thousand requests to be matched that were previously discarded in big cities such as Chengdu and NYC everyday. We propose exact algorithms to compute multi-hop trips, which achieves more than two orders of magnitude faster response time than the state-of-the-arts while improving the matching quality significantly. We further accelerate the matching efficiency by another order of magnitude using approximation strategies such as deep learning.

Our work supports the deployment of multi-hop ride-sharing in real-world scenarios and creates opportunities for many interesting future directions, e.g., price-aware or demand-aware multi-hop ride-sharing. It is also worth investigating more advanced deep learning models by considering real-time traffic conditions to predict more accurate reachable areas and achieve higher matching efficiency.

REFERENCES

[1] Shuo Ma, Yu Zheng, and Ouri Wolfson. T-share: A large-scale dynamic taxi ridesharing service. In *ICDE*, pages 410–421, 2013.
[2] James J Pan, Guoliang Li, and Juntao Hu. Ridesharing: simulator, benchmark, and evaluation. *PVLDB*, 12(10):1085–1098, 2019.
[3] Libin Zheng, Lei Chen, and Jieping Ye. Order dispatch in price-aware ridesharing. *PVLDB*, 11(8):853–865, 2018.
[4] Yongxin Tong, Libin Wang, Zimu Zhou, Lei Chen, Bowen Du, and Jieping Ye. Dynamic pricing in spatial crowdsourcing: A matching-based approach. In *SIGMOD*, pages 773–788, 2018.
[5] Yi Xu, Yongxin Tong, Yexuan Shi, Qian Tao, Ke Xu, and Wei Li. An efficient insertion operator in dynamic ridesharing services. In *ICDE*, pages 1022–1033, 2019.
[6] Yixin Xu, Jianzhong Qi, Renata Borovica-Gajic, and Lars Kulik. Geoprune: Efficiently finding shareable vehicles based on geometric properties. In *SSDBM*, 2020 (In press).
[7] Yan Zhao, Kai Zheng, Yue Cui, Han Su, Feida Zhu, and Xiaofang Zhou. Predictive task assignment in spatial crowdsourcing: a data-driven approach. In *ICDE*, 2020.
[8] Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences (PNAS)*, 114(3):462–467, 2017.
[9] Neda Masoud and R Jayakrishnan. A real-time algorithm to solve the peer-to-peer ride-matching problem in a flexible ridesharing system. *Transportation Research Part B: Methodological*, 106:218–236, 2017.
[10] Florian Drews and Dennis Luxen. Multi-hop ride sharing. In *Annual Symposium on Combinatorial Search (SoCS)*, 2013.
[11] Wesam Herbawi and Michael Weber. Modeling the multihop ridematching problem with time windows and solving it using genetic algorithms. In *IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, volume 1, pages 89–96, 2012.
[12] Wesam Herbawi and Michael Weber. Evolutionary multiobjective route planning in dynamic multi-hop ridesharing. In *European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP)*, pages 84–95, 2011.
[13] Man Lung Yiu, Nikos Mamoulis, and Dimitris Papadias. Aggregate nearest neighbor queries in road networks. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):820–833, 2005.
[14] Ishan Jindal, Tony, Qin, Xuewen Chen, Matthew Nokleby, and Jieping Ye. A unified neural network approach for estimating travel time and distance for a taxi trip. 2017.
[15] Jianzhong Qi, Wei Wang, Rui Zhang, and Zhuowei Zhao. A learning based approach to predict shortest-path distances. In *EDBT*, pages 367–370, 2020.
[16] Jean-François Cordeau and Gilbert Laporte. The dial-a-ride problem: models and algorithms. *Annals of Operations Research*, 153(1):29–46, 2007.
[17] Yongxin Tong, Yuxiang Zeng, Zimu Zhou, Lei Chen, Jieping Ye, and Ke Xu. A unified approach to route planning for shared mobility. *PVLDB*, 11(11):1633–1646, 2018.
[18] Lu Chen, Qilu Zhong, Xiaokui Xiao, Yunjun Gao, Pengfei Jin, and Christian S Jensen. Price-and-time-aware dynamic ridesharing. In *ICDE*, pages 1061–1072, 2018.
[19] Mohammad Asghari, Dingxiong Deng, Cyrus Shahabi, Ugur Demiryurek, and Yaguang Li. Price-aware real-time ride-sharing at scale: an auction-based approach. In *SIGSPATIAL*, page 3, 2016.
[20] Yunfei Hou, Xu Li, and Chunming Qiao. Tictac: From transfer-incapable carpooling to transfer-allowed carpooling. In *IEEE Global Communications Conference (GLOBECOM)*, pages 268–273, 2012.
[21] Brian Coltin and Manuela Veloso. Ridesharing with passenger transfers. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3278–3283, 2014.
[22] Zhou Zhao Da Yan and Wilfred Ng. Efficient algorithms for finding optimal meeting point on road networks. *PVLDB*, 4(11):1–11, 2011.
[23] Bin Yao, Zhongpu Chen, Xiaofeng Gao, Shuo Shang, Shuai Ma, and Minyi Guo. Flexible aggregate nearest neighbor queries in road networks. In *ICDE*, pages 761–772, 2018.
[24] Liang Zhu, Yinan Jing, Weiwei Sun, Dingding Mao, and Peng Liu. Voronoi-based aggregate nearest neighbor query processing in road networks. In *SIGSPATIAL*, pages 518–521, 2010.
[25] Peng Cheng, Hao Xin, and Lei Chen. Utility-aware ridesharing on road networks. In *SIGMOD*, pages 1197–1210, 2017.
[26] Yuxiang Zeng, Yongxin Tong, and Lei Chen. Last-mile delivery made practical: An efficient route planning framework with theoretical guarantees. *Proc. VLDB Endow.*, 13(3):320–333, 2019.
[27] Ashutosh Singh, Abubakr O Alabbasi, and Vaneet Aggarwal. A distributed model-free algorithm for multi-hop ride-sharing using deep reinforcement learning. *arXiv: Learning*, 2019.
[28] https://outreach.didichuxing.com/research/opendata/en/.
[29] Tlc trip record data, 2017.