2017 IEEE 33rd International Conference on Data Engineering

Xhare-a-Ride: A Search Optimized Dynamic Ride Sharing System with Approximation Guarantee

Raja Subramaniam Thangaraj*, Koyel Mukherjee†, Gurulingesh Raravi‡, Asmita Metrewar§,
Narendra Annamaneni* and Koushik Chattopadhyay*
*Conduent Labs India, Bangalore, India. †IBM Research-India, Bangalore, India.
‡CISTER Research Center, Porto, Portugal. §Apple India, Hyderabad, India.
Email: rajasubramaniam.t@xerox.com, kmukherj@in.ibm.com, guhri@isep.ipp.pt, asmita_metrewar@apple.com,
narendra.annamaneni@xerox.com and koushik.chattopadhyay@xerox.com

***Abstract*—Ride sharing is a sustainable, environmentally-friendly mode of commute that is gaining in popularity. Though there are well-established commercial service providers, there are not many platforms for facilitating peer-to-peer ride sharing, especially in a dynamic scenario, integrated with multi-modal trip planners. Such systems would need to be highly search-optimized for retrieval of multiple potential ride matches in real time, especially because multi-modal trip planners have a high look-to-book ratio. At the same time, validity of the matches need to be ensured, even in a dynamic setting, while addressing quality considerations and constraints such as maximum detour incurred by rides, walking distance for commuters, and time windows of requests. We describe Xhare-a-Ride (XAR) system, a platform for dynamic peer-to-peer ride sharing, that is scalable, efficient, and highly search-optimized for retrieving multiple potential matches for every ride request, while handling quality considerations. We propose a hierarchical discretization of the geographical region using grids, landmarks and clusters with theoretical guarantees, along with an efficient in-memory indexing of rides for maintaining spatio-temporal validity within a specified error tolerance. This helps eliminate shortest path computation in real-time during search, thus making XAR search-optimized, hence, suitable for integration with a multi-modal trip planner. We discuss modes of integrating XAR with such a trip planner for building an integrated system. Finally, we evaluate XAR thoroughly on ride share request data generated from the NY taxi trip data set on three fronts: (i) empirical performance against the theoretical guarantees as well as trade-off of performance with system parameters; (ii) benchmark XAR against a state-of-the-art ride share system, showing a significant improvement in the search efficiency; and finally, (iii) the efficacy of combining ride sharing with public transportation.**

I. Introduction

Ride sharing is gaining in popularity across the world with increasing traffic congestion, and rising fuel prices. It is convenient and faster compared to availing public transportation alone. On the other hand, it costs much less and is much more environment-friendly than driving one's own car with single occupancy. City planners and corporate organizations are encouraging ride-sharing as it reduces the number of vehicles on the road (for example, see [1] and [2]).

In fact, ride sharing can potentially alleviate the first and last mile connectivity problem for public transportation, and encourage their wider adoption. This would be of significant interest to city planners and governments as it would lead to more efficient utilization of the public resources and would potentially, lower the road congestion and carbon footprint, thus making such a multi-modal transportation highly sustainable and environment friendly. However, there are not many platforms that enable such an integrated multi-modal transport with ride sharing as one of the legs.

While commercial service providers ([3], [4]) have their own mobile and web applications through which they accept requests, and convey matches etc., the options for peer-to-peer ride sharing are limited and to the best of our knowledge, no such system is integrated with multi-modal trip planner. Though there exist platforms [5] that provide a matching platform to riders and drivers for carpooling, and some informal matches happen through social network etc., there is no established *dynamic peer-to-peer* ride sharing platform.

The Xhare-a-Ride (XAR) system is one of the first dynamic, scalable ride sharing systems for peer-to-peer ride sharing, that is highly search optimized while ensuring accuracy and quality considerations and constraints such as detour and walking preferences. It is also designed to return multiple ride options for each request. XAR can be easily integrated with a multi-modal trip planner because of the above characteristics.

We propose a *three-tiered* hierarchical region discretization, using grids, landmarks and clusters unlike the state-of-the art systems that mostly use grid-based region discretization. The hierarchical representation with (implicit) grids at the lowest level, landmarks on top of grids, with sets of grids mapping to landmarks based on maximum distance from a grid to a landmark, and finally clusters of landmarks, with bounds on the maximum distance between any two landmarks within a cluster helps to maintain ride information at the highest level in terms of clusters. State-of-the art dynamic ride share systems like T-Share [6] store the region information in terms of grids only, hence require shortest path computation in real-time, or, would need to store the pre-computed shortest path in the memory. In addition to this, spatio-temporal ride availability information has to be maintained at grid level. This would require a huge storage, that is either not available in a real-time system, otherwise the grid sizes would need to be large, thereby hindering the accuracy of such a system. We study the associated optimization problem in terms of hardness and give an algorithm for the region discretization with constant approximation guarantee.

On top of the discretization, we propose an efficient *in-memory indexing*, using clusters as the main units, that allows maintaining valid spatio-temporal information of rides and request dynamically within a maximum error tolerance in

This work was done when the authors were at Xerox Research Center India (currently known as Conduent Labs India).

2375-026X/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDE.2017.156

terms of the distance. The indexing eliminates shortest path computation for real-time searches, unlike existing systems [6], and helps in returning multiple valid matches for a given request. Note, the shortest path computation is required when a ride offer is created and when a request is actually booked on a chosen ride.

We evaluate the performance of XAR on real-world data through extensive experiments. Firstly, we evaluate it against theoretical guarantees, and examine the trade-off with system parameters. Then, we thoroughly benchmark XAR against T-Share. The benchmarking shows that while XAR is comparable to T-Share for ride creation and booking operations, it hugely outperforms T-Share for ride search operations, making it more suitable as a dynamic ride sharing platform with a high look-to-book ratio, where several potential matches can be returned for every ride request in real-time. In fact, we show empirically that XAR outperforms T-Share significantly as the look-to-book ratio of a system increases.

Another contribution of the XAR system is definition of systematic modes of interaction with a multi-modal trip planner (MMTP) [7] for integrating ride sharing with it. Multi-modal trip planning is commonly used by commuters[1], however, the suggested trip plans may include long walking distance, or waiting time or multiple hops or transfers. Observe that ride sharing may potentially help alleviate such first and last mile connectivity problems associated with public transportation, however, this would require planning trips using multiple modes of transport. MMTPs generally have a very high look-to-book ratio, hence, to integrate ride sharing effectively, the search operation has to be highly optimized. We show how to use XAR to make multi-modal trip plans more comfortable for users and we also address user experience and report quality metrics (in terms of user convenience) of the ride share and trip plan solutions.

A. Our Contributions

Our main contributions are as follows.

1) A three-tiered region discretization with efficient in-memory indexing for maintaining, updating and efficiently retrieving ride/request spatio-temporal information. This allows for optimized and scalable ride search in real-time, within an additive approximation guarantee on the total detour and walking distances.
2) Algorithms with theoretical analysis and proven constant approximations for the optimization problem associated with the region discretization problem.
3) Systematic modes of integration of a ride sharing platform with a multi-modal trip planner.
4) Extensive experimentation on NY taxi data [8], evaluating the performance of XAR against the worst case guarantees, benchmarking the performance of XAR against one of the state-of-the art systems, viz., T-Share, and evaluation of XAR integrated with OpenTripPlanner on NY GTFS data along with taxi data.

The paper is organized as follows. In the next section, we compare our work with the related literature and state-of-the-art. In Section IV, we describe the region discretization, and give theoretical analysis with hardness of approximation and constant approximation guarantees on the underlying optimization problem in Section V. In Section VI, we describe the in-memory indexing, We highlight the optimized search operation in Section VII, and then in Section VIII, we describe the update methods: tracking and booking. In Section IX, we describe the modes of integration with a multi-modal trip planner. Section X describes the experimental results and finally, we conclude in Section XI.

[1]Google Maps: https://www.google.com/maps.

II. Related Work

Ride sharing has been well-studied in the literature as well as by practitioners. Furuhata et al. [9] provide a comprehensive survey of the different types of ride sharing and associated problems studied. Ma et al. [6] study a dynamic taxi dispatching system for ride sharing. The objective is to efficiently search and reduce the overall distance traveled. They propose a spatio-temporal indexing to retrieve candidate taxis that are likely to satisfy a user query, however, they grid the underlying geographical region, unlike our hierarchical discretization approach. Moreover, they invoke a "lazy shortest path" calculation when required, to find candidate taxis that can make the search inefficient for a large-scale, dynamic ride share system. In contrast, we propose in-memory data structures, and efficient update methods that do not require computation of shortest paths in real-time for a search operations.

Huang et al.[10] study large scale real-time ride sharing, where they match trip requests dynamically to vehicles while satisfying waiting and service time constraints. They propose two scheduling algorithms using branch-and-bound and integer programming (not applicable in the dynamic scenario), and then give kinetic tree algorithms that are more efficient at scheduling dynamically and adjusting routes on-the-fly. Our work is complementary to theirs. We provide a search optimized ride sharing system, where valid search results or supply options corresponding to a ride request, satisfying detour, walking and time related constraints of all parties, can be quickly found by our system, and the actual re-routing of supply matched to demands dynamically can be done by a system as proposed by Huang et al.

Cao et al. [11] propose a dynamic ride sharing system that allows riders to specify their price, and internally computes the price of rides that are possible matches to ride requests based on the total ride distance and detour, and prunes the supply accordingly before returning search results. The system differs from ours in the following: only one rider can be matched to a driver, price is the constraint unlike detour and walking thresholds, and finally, it is not search optimized, but more optimized for pre-computing and returning a smaller set of rides such that the computation required for actually booking a ride is less.

Geisberger et al. [12] study the problem of computing detours efficiently while matching ride requests to trips. Pelzer et al. [13] study the problem of minimizing inconveniences to passengers due to ride sharing, while not letting the detours exceed a certain threshold. They achieve this by dividing the search space into partitions whose shape and size depend on the road topology. Though we maintain detour thresholds as well as partition the underlying geographical space, our discretization technique is very different and we propose new data structures for optimizing search efficiency.

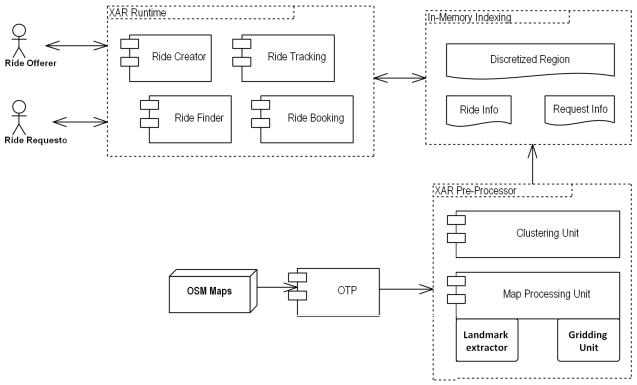


Fig. 1: The core components of the proposed XAR system.

Agatz et al.[14] provide optimization methods for minimizing the total system-wide vehicle miles. Ota et al. [15] propose a data-driven simulation framework that enables the analysis of a wide range of ride-sharing scenarios, and evaluate their simulation using NY taxi data. Santos et al. [16] propose a dynamic ride sharing system with time windows where the objective is to maximize the number of shares. Agatz et al. [17] provide a comprehensive survey of the optimization challenges and considerations, and existing research on dynamic ride sharing systems. They also talk about multi-modal trip planning with ride sharing. However, the problem that is referred by [17] and studied by [18] as multi-modal ride sharing is to find hubs or spots of heavy demand, and determine fixed route shuttle services to act as feeder routes to the main public transportation system. On the other hand, XAR provides a platform for dynamic, peer-to-peer ride sharing integrated with an MMTP. [17] also talk about integrating MMTP with capability to make taxi reservation on-demand, again quite different from ours, as studied by Lee et al. [19] and Li and Quadrifoglio [20].

III. System overview

The core components of XAR are shown in Figure 1. XAR mainly comprises of three components: XAR pre-processing unit, XAR run-time unit and XAR in-memory indexing.

XAR requires pre-processing of the geographical region where the system will be used. For example, if the region is a city, the entire city needs to be discretized into three tier hierarchical representation of grids, landmarks and clusters. The map-processor module in the pre-processing unit uses geographic information of the city from OpenStreetMap (OSM)[21] to create grids and extract landmarks. Further, these landmarks are grouped together into clusters by the clustering module, using route information from OpenTripPlanner (OTP)[22]. This pre-processing needs to be done once before deploying the system for each region. Complete details on hierarchical discretization of a region are given in Section IV and Section V. XAR run-time unit kicks in when the system is actually deployed. This unit provides functionalities like creating a ride offer in the system, finding matching rides for a request, booking seats in a matched ride and tracking rides in progress in real-time. Search functionality is elaborated in Section VII whereas tracking and booking functionalities are explained in Section VIII. XAR in-memory indexing stores all the relevant information in the system. It stores information about the discretization of the city such as grids, landmarks, clusters, distances between landmarks, etc. It also maintains information regarding rides offered and requests raised for shared rides. This component is explained further in Section VI.

IV. Hierarchical three-tiered Region Discretization

In this section, we describe the hierarchical discretization of geographical region. Traditionally, state-of-the-art ride sharing system implementations, such as T-Share [6] use a grid based representation of the underlying geographical region. The main difficulties of grid based representation are as follows. Such systems are not scalable in real-time for large regions, as they involve shortest path calculation in real-time. The scalability may be ensured by having larger grids, however that would lead to loss of accuracy in ride searching and matching, leading to invalid matches and inconsistencies in the travel time estimates, and current systems do not provide any worst case guarantees on the quality of matches. One may propose to handle the scalability by pre-computing all shortest paths, however, such a huge amount of storage is not generally available for real-time dynamic systems, and would involve querying the database at the expense of turnaround time. Hence, for a truly dynamic large scale ride share system, we need a region discretization that is efficient and scalable and does not involve shortest path calculation, and at the same time, is accurate enough to prevent invalid ride matches and/or incorrect travel time estimates within some error tolerances.

We propose a hierarchical, three-tiered discretization, where the three entities used for the representation of a region are: clusters, landmarks and grids. The well-defined, systematic representation ensures that any point location, given by a latitude and a longitude can be uniquely mapped to a grid, then a landmark and finally a cluster, *without any ambiguity*. The hierarchy is as follows: region $\rightarrow$ clusters $\rightarrow$ landmarks $\rightarrow$ grids $\rightarrow$ point locations, however, at the same time, there are cross relations across the levels, where grids are directly associated with clusters. This will be explained in the following.

Definition 1: A grid is defined as a bounded square geographical region. All point locations whose latitude and longitude map to the region bounded by the square defining a grid, are associated or mapped to the specific grid.

From the definition of a grid, it is clear that the relation between point locations and grids are many-to-one. In other words, any point location maps to a unique grid. Any location is numerically mapped to a unique (implicit) grid, given its latitude and longitude. Since in our application, we consider very small grids of size $100\ m^2$, for all practical purposes, we identify a grid by its centroid, and henceforth wherever we refer to distances from a grid, these are measured with respect to the centroid of the grid.

Definition 2: A landmark is a point of interest in a geographical region, such as a bus stop, a mall or an important building, such that it is sufficiently far (at least a pre-specified f distance away) from any other landmark.

Each landmark is associated with a set of grids, and each grid is associated with a unique landmark, that minimizes the maximum driving distance of the grid from the landmark. If

there are multiple such landmarks, we choose the one with the lowest number in an ordering imposed on the set of landmarks. For inhabited regions, each grid will have at least one landmark within a certain Δ driving distance of itself with a high probability, where Δ is a pre-specified system parameter. We will associate a grid with a landmark only if it is within Δ distance of the landmark. For remote locations, or due to one-ways or road blocks or narrow lanes etc., it is possible that the nearest landmark in terms of driving distance from a grid is farther than Δ distance away. In this case, such a grid will not be associated with any landmark, however, can still be referenced by the system in terms of clusters that are within walking distance from the grid, as discussed next.

Definition 3: A cluster is defined as a collection of landmarks, such that no pair of landmarks in a cluster are more than a specified δ driving distance away.

As already stated, along with driving distance, it is important to maintain the record of walking distance of a grid from the landmarks and clusters, as the two can sometimes be very different, especially in regions with narrow streets, or one-way etc. Hence, with each grid, apart from the nearest landmark information, we additionally maintain a list of "walkable clusters". Walkable clusters of a grid are maintained as a list of tuples: $< C, w >$, where w is the distance of the nearest landmark in the cluster C from the grid, and $w \leq W$, where W is another system parameter defining the maximum walking distance allowed by the system. The list is sorted in non-decreasing walking distances of the clusters from the grid. Informally, this is a list of clusters that are within a walking distance W of the grid. The actual walking distance w corresponding to any walkable cluster C (this is the distance of the grid from the nearest landmark in C) is precomputed and stored. While actually serving a request, the list of walkable clusters can be further pruned according to the walking distance threshold mentioned by the commuter for a particular trip request, by traversing this sorted list in time linear in the number of walkable clusters.

Hence, even if a grid is not associated with any landmark due to the driving distance, with a high probability, it will be within walking distance of some landmarks of some clusters for grids corresponding to inhabited regions. If a grid is neither in the driving distance of a landmark, hence cluster, nor within the walking distance of any landmarks/cluster, then requests from it will not be served. It is important to note that a 'cluster' does not define any bounded particular geographical region. It is simply defined as a collection of landmarks, such that any ride requester can be asked to walk to any of these landmarks, and any ride can be routed to pick a requester up from any of these landmarks for valid matches satisfying all other constraints. This definition of clusters in terms of landmarks is crucial in enabling the storage of information regarding rides dynamically, keeping track of changing parameters of rides.

The landmarks are extracted from a multi-modal trip planner or a map such as OpenStreetMap [21]. The first step is to filter the set of landmarks to ensure that no two landmarks are located too close ($< f$ distance, where f is a system parameter) to each other, as that would defy the purpose. Once we have a distinct set of landmarks, such that each landmark is sufficiently far from all the other landmarks in the system, our problem is to find a set of clusters such that in any cluster, no two landmarks are too far away, in other words, within a δ distance of one another, where δ is another system parameter. At the same time, every landmark should be associated with a distinct cluster, and the number of clusters stored should be minimized, as otherwise, the efficiency of search query would be affected and we want to optimize the storage required in the RAM. We denote this problem as CLUSTERMINIMIZATION, formulate it as an integer linear program, and study it analytically in Section V. Specifically, we show that the problem is hard to approximate and give a simple greedy algorithm with a bicriteria approximation for finding the clusters in a geographical region.

V. FINDING THE CLUSTERS

The problem CLUSTERMINIMIZATION can be written as an integer linear program as follows. In the following program, we denote the set of (filtered) distinct landmarks as $\mathcal{V}$, where $|\mathcal{V}| = n$. We use indicator variable $x_{i,j}$ to denote if landmark i is mapped to the j^{th} cluster. The number of clusters used is denoted by m, and indicator y_j denotes whether j^{th} cluster is used. Note that in the limiting case, there can be at most n clusters, where each landmark is its own cluster.

$$
\begin{aligned}
&\text{Minimize } m \\
&\text{such that } \sum_{j \in [n]} y_j \leq m \\
&\qquad x_{i,j} \leq y_j \quad \forall i \in \mathcal{V}, j \in [n] \\
&\qquad \sum_{j \in [n]} x_{i,j} = 1 \quad \forall i \in \mathcal{V} \\
&d_{i,i'}(x_{i,j} + x_{i',j} - 1) \leq \delta \quad \forall i, i' \in \mathcal{V}, \forall j \in [n] \\
&\qquad x_{i,j} \in \{0,1\} \quad \forall i \in \mathcal{V}, j \in [n] \\
&\qquad y_j \in \{0,1\} \quad \forall j \in [n]
\end{aligned}
$$

In the above program, the objective is to minimize the number of clusters used m. m is lower bounded by the sum of y_j over all $j \in [n]$, where y_j is an indicator variable denoting whether a cluster j is used or not. Since we want to minimize m, the program will set m to be equal to the sum of y_j's over all $j \in [n]$. This is captured in the first inequality. The second inequality says that a landmark can be associated with a cluster only if the cluster is used. The third equality states that every landmark must be associated with exactly one cluster. The fourth inequality states that if two landmarks are part of the same cluster, then their pairwise distance should be at most δ. Note that in case the distance between a pair of landmarks i and i' is greater than δ, this would force that for no j, can both these landmarks belong to j. Finally, we have the range specifiers.

Solving an integer linear program is computationally intensive, especially, given the large set of landmarks in a geographical region. Furthermore, we next prove that this problem is NP-hard, thus precluding the possibility of polynomial time algorithms for solving it optimally if $P \neq NP$. Hence, we look at approximations. The underlying problem setting can be represented as an unweighted graph, where landmarks are vertices, and an edge between any two vertices exist if and only if the distance between the corresponding landmarks is $\leq \delta$. Now, in this unweighted graph, the problem is equivalent to partition the graph in to min. number of cliques, such that every vertex is part of a clique.

Theorem 4: The problem of minimizing the number of clusters, referred to as CLUSTERMINIMIZATION subject to the

conditions that every landmark is part of at least one cluster and the pairwise distance between any two landmarks in a clusters is at most δ, is NP-complete.

The proof is omitted due to lack of space. We next show that getting a constant factor approximation to minimizing the number of clusters given a distance threshold δ is likely to be hard for certain metrics.

Theorem 5: There exists metrics where the CLUSTERMINIMIZATION problem is $(1-\epsilon)\ln n$ hard to approximate unless $NP \subset TIME^{n(O \log\log n)}$.

The proof is omitted due to lack of space. It follows from Theorem 4 and Feige's [23] result that if there exists a polynomial time algorithm that can approximate the SETCOVER problem within $(1-\epsilon)\ln n$ factor, then $NP \subset TIME^{n(O \log\log n)}$.

From Theorem 5, it is clear that approximating the number of clusters k, given a distance threshold δ, within any constant $c > 0$ is likely to be hard. Therefore, we next look at a bicriteria variant of the CLUSTERMINIMIZATION problem, where we guarantee the same number of clusters as the optimal solution, but slightly compromise on the distance threshold by stretching it within a constant factor.

We now describe an algorithm GREEDYSEARCH for approximating the bicriteria problem (k, δ) for CLUSTERMINIMIZATION on a metric space. We use the well-known 2-approximate greedy algorithm [24] for METRIC K-CENTER as a subroutine for getting an approximation to the CLUSTERMINIMIZATION problem (henceforth we refer to this subroutine as GREEDY). The main algorithm GREEDYSEARCH works as follows. Guess a value of k, $k \in \{1, n\}$, where $n = |\mathcal{V}|$, is the total number of landmarks. Now, call GREEDY to get k centers. Next, check the maximum distance of any landmark from any of the chosen k centers returned by the greedy algorithm. If the maximum distance of any landmark from its center is $> 2\delta$, where δ is the specified inter-landmark distance in any cluster, then binary search on the upper half of the current range of k. Otherwise, binary search on the lower half of the current range. Repeat for $\log_2 n$ iterations. The pseudocode is omitted due to lack of space. The algorithm returns $\log_2 n$ tuples of the form $(k', \delta_{k'})$, where δ_k is the maximum distance of any landmark from its center for the given choice of k. We choose the minimum value of k' for which $\delta_{k'} \leq 2\delta$ as k_{ALG}.

Theorem 6: Let the optimal solution to CLUSTERMINIMIZATION be (k_{OPT}, δ), where k_{OPT} is optimal number of clusters, and δ is the maximum distance between any two landmarks in a cluster. Then the solution produced by the GREEDYSEARCH guarantees the following bicriteria approximation: $(k_{OPT}, 4\delta)$. In other words, the number of clusters returned by the GREEDYSEARCH, $k_{ALG} \leq k_{OPT}$ and the maximum distance between any two landmarks in a cluster is at most 4δ.

Proof: Let us denote the k chosen in any iteration j as k_j. Suppose k_{ALG}, or the lowest value of k for which the distance of any landmark from its center is $\leq 2\delta$, occurs in iteration i, i.e., $k_{ALG} = k_i$.

This implies that for every k_j, $j > i$, the there exists at least one landmark such that its distance from its center is $> 2\delta$ in the solution returned by the greedy algorithm GREEDY. We know that GREEDY is a 2 approximation to METRIC K-CENTER. In other words, the maximum distance of any landmark to its center for the given k is at most twice that of any optimal solution. Thus, it implies that for any $k < k_i$, in any optimal solution to the METRIC K-CENTER problem for the given instance, there will exist at least one landmark such that its distance from its center is $> \delta$.

We next argue that an optimal solution to the CLUSTERMINIMIZATION problem on the given instance for the given δ requires $\geq k_i$ clusters. Suppose there exists an optimal solution with $k < k_i$ clusters. Now, lets compute a solution to the METRIC K-CENTER problem from the optimal solution to the CLUSTERMINIMIZATION problem on the same instance. From every cluster, arbitrarily choose any landmark as the center, and assign all the landmarks in the same cluster to this center, for the corresponding solution to the METRIC K-CENTER problem. But that gives a set of $k < k_i$ clusters, such that the maximum distance of any landmark from its center is at most δ, thereby giving a contradiction. Therefore, any optimal solution to the CLUSTERMINIMIZATION problem on the given instance, would require $\geq k_i$ clusters. Since $k_{ALG} = k_i$, this implies that $k_{ALG} \leq k_{OPT}$. Now, assign all the landmarks assigned to a center, along with the center landmark to a cluster to get k_{ALG} clusters. We know that the maximum distance of any landmark to its center is at most 2δ by the choice of k_{ALG}. From triangle inequality, it therefore follows that no two landmarks assigned to the same cluster are more than 4δ distance away. The bicriteria approximation factor therefore follows. ■

Given the bicriteria approximation, we set $\epsilon = 4\delta$ and ensure that no two landmarks in a cluster are more than ϵ away in the worst case. In the next section, we describe the in-memory indexing on top of hierarchical region discretization for storing the dynamically changing ride information that is enabled by such discretization. The indexing allows us to guarantee that in the worst case the detour limit of a ride will be exceeded by at most a 4ϵ additive factor, while we show later empirically, that for 98% of the cases, the detour limit is exceeded by at most an additive ϵ distance. The walking distance thresholds of requests are strictly met.

VI. INDEXING STRUCTURE FOR EFFICIENT SEARCHING

So far, we have discussed how we identify grids, landmarks and clusters. Now, in this section, we discuss how the information about the rides is organized/indexed in the XAR system using the region discretization to facilitate efficient searching.

Essentially, any ride sharing system such as XAR needs to effectively perform following operations: (O1) For a new ride request, it should efficiently identify those rides that can potentially serve the request (considering the constraints of the rider and the requester); (O2) For a new ride being offered, it should efficiently capture the route that the ride is taking and the surrounding area of that route so that when a new request arises along the route or in the surrounding area then operation O1 can be effectively performed; and (O3) For a ride that is in progress (i.e., ride has departed from its source and is on its way to destination), it should efficiently update its current location so that for a new request arising from the part of the route (or the surrounding area) that the ride has already passed, this ride should not be mistakenly shown as one of the potential rides (that would eventually lead to wrongly matching the request to the ride). So, information about rides

in the system should be organized such that the previously mentioned three operations can be effectively done, thereby facilitating efficient matching of requests to appropriate rides. We next outline the information storage structure deployed in the XAR system.

For any given ride, we maintain information regarding the clusters that the ride passes through and the clusters that the ride can reach without violating its detour limit, and this information is updated regularly, in order to ensure correctness of the information. Additionally, for each cluster, information about all the potential rides that can reach the cluster, along with their estimated time of arrival is maintained. Note that we use the universal representation of a route in terms of waypoints[2].

We now define the following entities that characterize a ride in XAR.

1) *source* location: the location where a ride begins its journey,
2) *destination* location: the location where a ride ends,
3) *departure time*: the time when the ride begins,
4) *seats*: number of seats available in the ride,
5) *route* of the ride: the shortest route between the source and the destination unless the user has explicitly specified an alternate route,
6) *via-points*: the point locations through which a ride passes; these point locations correspond to the pick-up/drop-off points of the co-riders including the source and destination of the ride itself. These are the set of geographical locations through which the ride necessarily must pass [3].
7) *segment*: the portion of the route between a pair of via-points,
8) *detour* limit of the ride: the deviation that the driver is willing to take from the route of the ride in order to serve the shared ride requests,
9) *pass through clusters*: the clusters through which the ride passes in a segment, and
10) *reachable clusters*: the clusters that the ride can reach without violating the detour limit.

Additionally, each cluster has a list of rides associated with it as *potential rides*. With each cluster C, this information is maintained as a list of tuples of the form: $< r, t >$, where r denotes a ride in the system, and t is the estimated time of arrival of the ride in the cluster C. We maintain the tuples in two different lists, one sorted in non-decreasing order by the time of arrival, and the other sorted by the unique ride identification numbers. Each ride created in the system is assigned a unique ride ID. The time of arrival is estimated from historical travel times. Figure 2 depicts some of these terminologies with an example. For a ride offered in the system, the grids through which the route of the ride passes are identified. Then by identifying the unique landmarks that are associated to each of these grids, we in turn identify the clusters in which these landmarks are present; these clusters constitute the *pass-through* clusters of the ride.

After determining the pass-through clusters, we determine the reachable clusters *associated with each pass-through cluster* as follows. Let the route of the ride be given by k via points, including the source and destination. Consider the segment between via-point i and $i+1$, for some $1 \leq i < k$. Let the set of pass-through clusters in this segment be $\mathcal{C}_i$. For each pass-through cluster $C \in \mathcal{C}_i$, first, find a set of candidate reachable clusters for C: this set is constituted of all clusters that are reachable within the detour limit d of the ride from C. (Note that the distance between clusters is determined by the distance between the closest pair of landmarks belonging to the two clusters, respectively). Let this candidate set of clusters be $\mathcal{C}_r$. Now, choose a cluster C' in $\mathcal{C}_r$. Let the distance from C to C' be $d_{C,C'}$, the distance from C' to via-point $i+1$ be $d_{C',i+1}$ and the distance from C to via-point $i+1$ be $d_{C,i+1}$. If $d_{C,C'} + d_{C',i+1} - d_{C,i+1} \leq d$, where d is the current detour limit, then we leave it in the set $\mathcal{C}_r$, otherwise we remove it from the set. Then we repeat the same with the next candidate cluster in $\mathcal{C}_r$, till all clusters in the set are examined. The resultant set of clusters $\mathcal{C}_r$ are associated with C as the *reachable* clusters from C.

The main idea is that any request originating in the reachable clusters of a pass-through cluster of a ride is a potential feasible match for the ride, and this can be determined dynamically too, while the vehicle is passing through the pass-through cluster. Once we identify the pass-through cluster and their associated reachable clusters for every segment of the route of a ride, the ride gets added to the list of *potential rides* associated with each of these clusters, maintaining the sorted order of the lists.

We next describe the search optimization enabled by the indexing described in the previous section built on top of the three-tiered region discretization. The search optimization is one of the main highlights of the XAR system and is crucial for a dynamic ride share system with high look-to-book ratio (especially so when integrated with an MMTP).

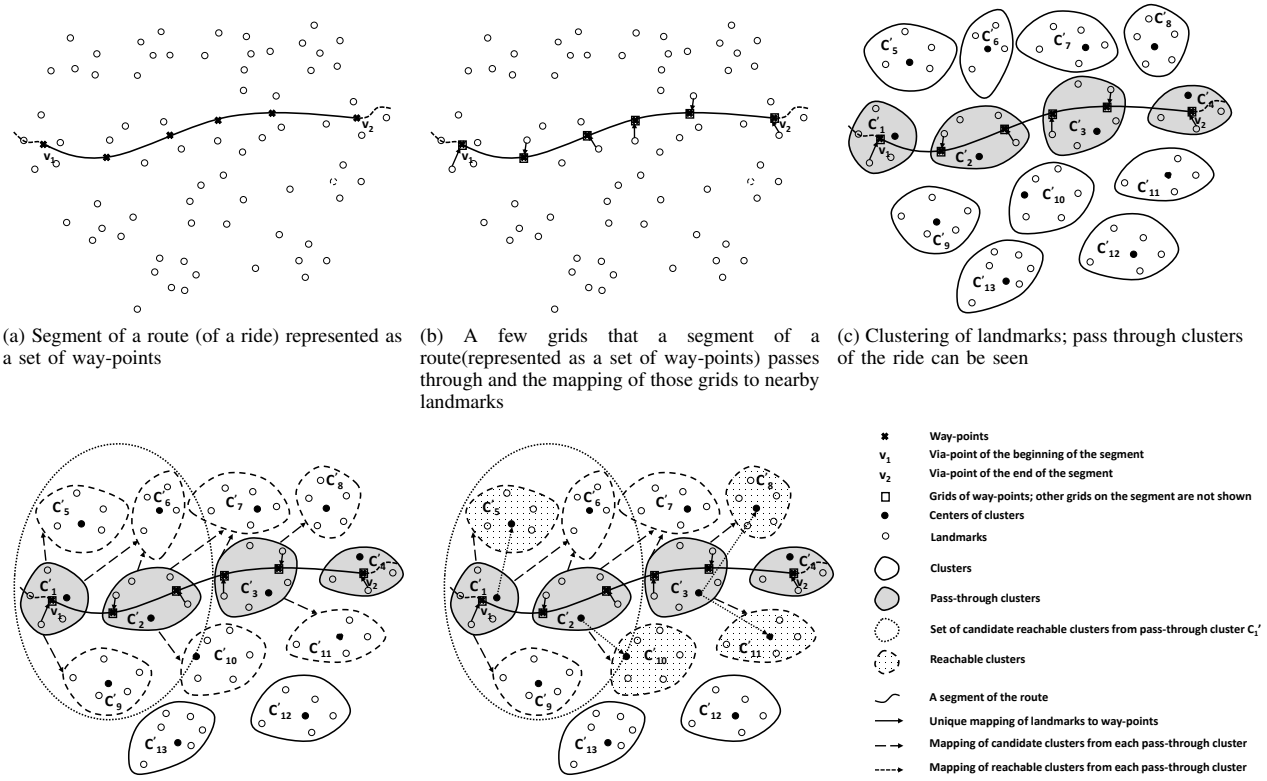
VII. Optimized Ride Search Operation

When a new ride request is raised, the system must efficiently extract only those rides that can potentially serve the request. A ride request is characterised by the following information: *source location*, *destination location*, *departure time window* and *walking threshold*. The departure time window is the window of time within which the request must be served, and the walking threshold is the maximum walking distance that the requester would like to incur in a match. Using this information, a set of relevant rides that respect all the constraints of both the ride offerer and the requester need to be identified and extracted.

Shortest path computations is one of the most expensive task of search operation in ride sharing system. Most of the existing systems perform shortest path computations during the search operation. This could work reasonably fine for systems which finds only one ride offer for each ride request. However, it will not scale when one wants to extract multiple ride offers for each ride request. We are interested in returning multiple ride matches for each request, as it enables the ride requester to choose an option suitable for her from the suggested options. For example, if a social networking graph could be built or integrated into the system then the rides offered by people in the social network graph of the requester can be given higher priority while listing the options. This will address the safety concern to some extent as people generally feel safe to travel with co-passengers from their social network.

[2]OpenStreetMaps represent the underlying road network as a graph where the vertices correspond to waypoints.

[3]Note that via-points are different from way-points.



(a) Segment of a route (of a ride) represented as a set of way-points

(b) A few grids that a segment of a route(represented as a set of way-points) passes through and the mapping of those grids to nearby landmarks

(c) Clustering of landmarks; pass through clusters of the ride can be seen

(d) Set of candidate clusters of pass-through cluster C_1'; candidate clusters of other pass-through clusters are not shown for ease of visualization

(e) Reachable clusters of each pass through cluster (obtained from the candidate clusters)

(f) Notations used in the example

Fig. 2: An example illustrating schematically a segment of the route, landmarks, clusters, way-points on a segment, via-points of the segment, pass through and reachable clusters

Further, when we integrate the ride sharing system to an MMTP as a mode of transport, there is generally a high look to book ratio, i.e., the number of searches/looks are much greater than the number of bookings. This further increases the computation load on the system and hence performing shortest path computations during search operation will deteriorate the performance of the system significantly. Therefore, we have consciously taken a decision while designing this system not to perform shortest path computations during the search operation.

With this background, we now describe the working of the ride search operation. It can be broadly described as a two-step procedure as follows.

Step 1. A set of candidate rides are identified as follows. Initially, let this candidate set be the empty set $\mathcal{R}_1$. We first identify the grid to which the source location of the request belongs. The set of *walkable clusters* associated with this grid, pruned further according to the walking limit specified by the requester for this particular trip request, are identified as the feasible set of clusters $\mathcal{C}$. Note that the pruning can be done efficiently by traversing in linear time (linear in the number of walkable clusters) on the sorted list of walkable clusters. Now, for each cluster $C \in \mathcal{C}$, we have a list of *potential rides*, $\mathcal{P}_C$. Now, we search in $\mathcal{P}_C$ to determine the set of rides whose estimated time of arrival in C matches the departure time window of the ride request. This can be done in logarithmic time on the sorted list. The corresponding rides are added to $\mathcal{R}_1$. This is repeated for each cluster in $\mathcal{C}$.

Step 2. Similar to Step 1, we find another set of candidate rides from the destination location, denoted as $\mathcal{R}_2$. The final set of candidate rides $\mathcal{R}'$ is formed by the intersection of these sets, i.e. $\mathcal{R}' = \mathcal{R}_1 \cap \mathcal{R}_2$

Finally, on the resultant set of candidate rides $\mathcal{R}'$, we do the following checks. For each ride $r \in \mathcal{R}'$, we check if the combined walking distance at the source and the destination locations is within the walking limit of the requester. Then, we check if the combined detour at the source and the destination locations is within the current detour limit of the ride. If any of the two condition fails, we remove r. Once we examine all rides, $\mathcal{R}'$ gives the set of feasible ride matches for the ride request.

VIII. Tracking and Booking

In this section we will focus on tracking and booking operations. Since our system is dynamic, the information regarding the current state, as in, the current routes and locations of active rides need to be maintained and updated in the system regularly, in an efficient and scalable manner. This is referred to as Tracking. Moreover, when a ride booking confirmation happens, it would also require updating the ride information in the system based on change in the route, the number of

seats available and the detour limit budget remaining. This is referred to as Booking.

A. Ride Tracking

Ride tracking operation essentially involves updating spatio-temporal status of a ride, from the time the ride has started its journey till its end. The information that mainly needs to be updated is the set of clusters that are associated to the ride as pass-through clusters and reachable clusters. Once the ride is on the move, it would have already crossed some of the clusters in the set and it cannot reach some of the clusters any more without violating its detour limit. Such clusters are collectively referred to as *obsolete* clusters for the ride. Hence, removing these obsolete clusters from the set of pass-though and reachable clusters of this ride is very essential for an accurate dynamic ride share system for matching new requests to the ride when the ride is on the move.

The method to update the data structures mainly consists of the following three steps. For each pass-through cluster that the ride has already crossed, do the following:

Step 1. Mark the pass-through cluster and all connected reachable clusters as obsolete for the ride.

Step 2. [Note that a cluster can be reached by the ride from multiple pass through clusters without violating detour limit.] For each cluster marked as obsolete for the ride, check whether the cluster can be reached through any of the valid pass-through clusters; if not, remove the ride from the list of potential rides for the cluster.

Step 3. Remove all the pass-through clusters marked as obsolete from the list of pass-through clusters.

The second step keeps the list of potential rides for the clusters up-to-date, while the third step keeps list of pass-through clusters of the ride up-to-date.

B. Ride Booking

When a ride requester confirms/books a shared ride, the route information needs to be updated in order to maintain accurate spatio-temporal ride information, the detour limit of the ride need to be updated depending on detour experienced by the ride in serving this booking and finally via-points and thereby new segments need to be created at the source and destination locations of the request. Finally, the pass through clusters and the reachable clusters of the ride need to be updated depending on the new route and its new segments and the new detour limit of the ride. Such an update may render some of the earlier pass through and reachable clusters invalid (due to change in route and detour limit).

Updating the route information will require shortest path computations. We avoided computing shortest paths during search operation. However, we perform shortest path computations once the booking is confirmed as a back-end process, thereby not compromising on the end user experience. Let us now discuss the steps to update.

New via-points (and hence segments) are created on the route of the ride corresponding to the source and destination location of the request. Let these via-points be denoted as src and $dest$.

Step 1. Compute the segments on which src and $dest$ lie. Let these segments be denoted s and d respectively.

Step 2. If both src and $dest$ lie on the same segment (say $s = d$) of the current route of the ride, then the shortest paths between starting point of segment s and src, between src and $dest$, and between $dest$ and end point of segment s need to be computed. The route of the ride is accordingly updated.

Step 3. If the source and destination locations of the request do not lie on the same segment of the ride then, let s_1 and s_2 be the start and end via-points of segment s on which src lies and d_1 and d_2 be the start and end via-points of segment d on which $dest$ lies. Now, shortest paths between s_1 to src, and src to s_2, as well as shortest paths between d_1 to $dest$ and $dest$ to d_2 are computed and the route is accordingly updated.

As can be seen from the above steps, in the worst case, 4 new shortest path computations need to be performed for every booking. However, as described earlier, since it is done in the back-end after the booking is confirmed, it does not affect the user experience.

We next describe the modes and associated methods of integration of XAR with any multi-modal trip planner.

IX. Integration with Multi-modal Trip Planner

So far, we have described the XAR system in detail, that can be used as a stand-alone ride share application. However, ride share can be even more effective if it is integrated as a leg of a multi-modal trip plan. While MMTPs and ride sharing platforms exist in isolation, there is no system currently that integrates both. We next describe systematic modes of integration of XAR with an MMTP.

A. Aider Mode

In aider mode, XAR module aids MMTP by providing shared ride options for any infeasible segment of the trip. A segment is considered infeasible if it incurs a long waiting time, walking distance, etc. than what is acceptable to the commuter. If the trip plan generated by the MMTP has an infeasible segment then MMTP checks XAR for the availability of shared ride options for the infeasible segment by providing source, destination and time interval in which the ride should begin. The source and destination may not be the original source and destination of the commuter but that of the particular infeasible segment of the trip.

B. Enhancer Mode

In enhancer mode, XAR module tries to enhance the trip plans generated by an MMTP. MMTP provides entire trip plan that it generated and asks XAR to enhance it. XAR checks for shared ride options for any segment of commuter's journey that can improve the quality of journey, specifically, reducing the number of hops or the travel time etc. If XAR finds a shared ride option that can enhance the trip plan by replacing one or more segments of the trip plan then XAR composes the enhanced trip plan and sends to the MMTP.

In a given trip plan, if the number of intermediate hops is $k \leq 4$, then XAR generates search requests for $\binom{k+1}{2}$

combinations of intermediate hops and source and destination[4], in order to replace these segments by shared rides. In case the number of intermediate points is > 4, which is extremely unlikely in a trip plan, then we will only consider the segments resulting from combinations of source to intermediate points, and intermediate points to destination, and not those between pairs of intermediate points. This results in $2k + 1$ segments to consider, including the entire journey, which is linear in the input size. This trade-off is done to ensure that the search operation for a particular trip request is completed within a reasonable amount of time, so that the user experience does not suffer when there is a lot of requests generated simultaneously. We aim to keep the enhanced search for one commuting request under 50 ms, such that even if there are 200 trip requests generated simultaneously, the total turn over time remains under 10 secs.

For each segment, we see if a shared ride exists with the source and destination for the particular segment, satisfying the time constraints. In other words, the ride should start only after the commuter would reach the intermediate source and should reach the intermediate destination before the next leg of the journey starts as per the current trip plan. Moreover, it should be feasible both for the commuter and the driver in terms of detour/walking thresholds. If there exists such ride options, these are returned to the end user, enhancing the trip plan generated by the multi-modal trip planner. Clearly, these trip plans will reduce the number of hops, and in case, a ride share is found to substitute the entire journey, it would potentially also reduce the travel time, thus resulting in more convenient options for the commuter.

X. Experiments

In this section, we present the observations from experimental evaluation performed on XAR system using real-world data. We perform three sets of experiments to evaluate XAR. In the first set, we evaluate the quality of ride sharing attained through ϵ additive approximation guarantee on the detours. In the second set, we look at how our system fares as compared to the current state-of-the-art dynamic ride sharing system, viz., T-Share [6] in terms of time taken for ride sharing operations. Finally, we throw light on whether ride sharing and public transport together can provide an environment-friendly travel option with better travel quality. Before getting into details of these experiments, we first describe the real-word data set used in these experiments and the framework for simulation.

A. Experimental Setup

1) Data set: For all the experiments, we have used the publicly available New York city taxi trip data [8]. Every trip in the dataset has a pickup time, a pickup location and a drop-off location. We randomly selected a day (March 7th, 2013), and extracted taxi trips for the day from this dataset. It had approximately $350,000$ taxi trips. In the original data set, these are individual taxi trips. We next describe how we generate ride sharing data from the taxi data.

[4]There are $\binom{k+2}{2}$ combinations of the $k + 2$ points, including source, destination and k intermediate points. However, we don't want to consider pairs of adjacent points in the current trip plan, hence the number of combinations is coming out to be $\binom{k+1}{2}$, which is at most 10 combinations for $k \leq 4$.

2) Simulation Framework: In order to evaluate XAR on the NY taxi data, we consider all the trips in the data set as requests for sharing rides. For simulating ride sharing, we iterate through the requests and for each request, we first try to search for an existing ride which could be matched with this ride request. If a ride is found, this request is matched with the ride found, thus, booking it. If multiple potential rides are found, the ride that incurs least walking for the requester is matched and booked. If no such rides are found, a new ride is created from this request and made available to be shared. Taxi capacity is assumed to be 4 (including the driver).

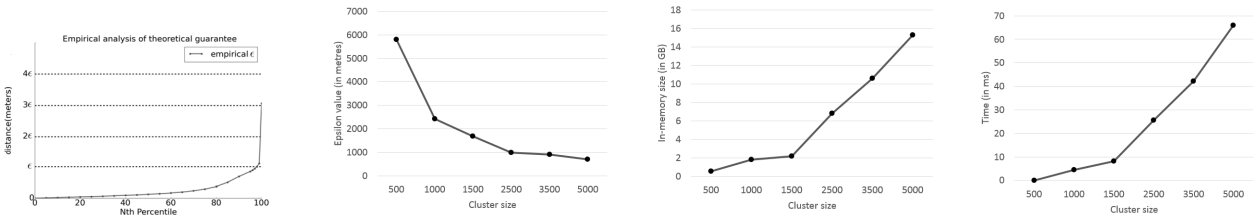
XAR is implemented in Java as a single-threaded standalone application and all the experiments are executed on a machine with 2.4GHz processor and 128GB RAM. Classmexer Java instrumentation agent [25] is used to measure the size of in-memory data structure. OpenTripPlanner [22], hosted as a web service, serves as an MMTP to obtain trip plans of public transport as well as a routing engine to get a driving route.

3) Region Discretization: We gridded NY city using map from Google, with each grid being $\approx 100m^2$ in size. For extracting landmarks, we created a temporary gridding with the grid size being $500m^2$ and queried landmarks using Google Places APIs [26] within the temporary grid. This gave around 30000 unique landmarks in the city. This list of landmarks was pruned to remove insignificant landmarks (e.g., small stores), leading to a final list of 16000 landmarks which include *bus stops, railway stations, big stores, taxi stands*, etc. We then applied the GreedySearch algorithm for clustering on this list to form clusters with a worst case distance guarantee of ϵ between any pair of landmarks in this cluster. In all the experiments, except those for measuring the quality of matching rides, $\epsilon = 1$km.

B. Experimental Results

1) Quality of Matching Rides: In this section, we empirically evaluated the theoretical approximation guarantee. In section V, we have proved that the theoretical upper bound on the driving distance between any two landmarks within a cluster is ϵ. Hence, any request matched to a ride will have a theoretical worst case detour approximation of 4ϵ. We evaluated the detour approximation value for $350,000$ requests and measured how well the system performs compared to the theoretical upper bound guarantee. From Figure 3a, it can be seen that 99.9% of the request matches have approximated detour less than 2ϵ while 98% are less than ϵ. This proves that in practical scenarios, XAR system performs much more efficiently than what is theoretically proven.

We also study how varying ϵ affects the number of clusters ($\mathcal{C}$), the memory requirements of the proposed indexing structure, and the performance of search operation. To stress test our system performance, we performed these evaluations against $120,000$ ride offers and $350,000$ ride requests. Figure 3 shows that ϵ and $\mathcal{C}$ are inversely proportional. Increasing $\mathcal{C}$ reduces ϵ, thereby increasing accuracy, but at the cost of performance and resources. At $\mathcal{C} = 500$, in-memory structure needs very less space and processing time is less than a millisecond, but the approximation guarantee (ϵ) is very high. In contrast, at $\mathcal{C} = 5000$, ϵ is just 700m but the memory needed to hold the in-memory index is almost 16GB and the processing time of ride search is 65ms for one request. Thus, ϵ can be tuned in order to trade-off between accuracy of the solution, and the desired performance and required scalability.



(a) Approximated detour for ride requests (b) Number of clusters as ϵ changes (c) Size of in-memory index (in GB) (d) Ride search time (in ms)

Fig. 3: Performance vs Approximation trade-off

2) Comparison with State-of-the-art: In this set of experiments, we compare XAR with the current state-of-the-art, viz., T-Share [6]. Specifically, the performance of our system is compared with that of T-Share in terms of time taken to serve the queries for searching, creating and booking shared rides. We extracted a subset of $100,000$ trips from the aforementioned data set, requesting pick-ups between 6am - 12pm. T-Share mentions the size of a grid as a configurable parameter. In the experiments, we have set the grid size to be $1000m^2$ (equivalent to the cluster size of XAR). [5]

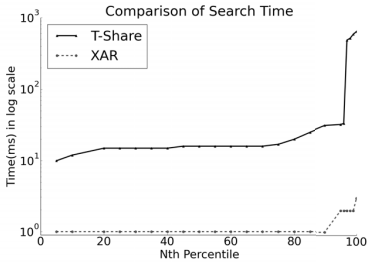
In the first experiment in this set, we compare the average case performance of both the systems in terms of time taken to search, create and book rides on 20000 rides and $100,000$ requests. We run the simulation on T-Share and XAR to find all the possible matches for a given ride request. In its original form, T-Share stops searching the region as soon as it finds one match for the request, and hence, returns only one match. We updated the matching module of T-Share implementation to search the region until it finds all the taxis in the region which can be matched with the request. Besides, T-Share keeps searching the space until it finds a match, exploring the entire search space in the worst case just to find a single match. Since the aim of this experiment was to compare performance in terms of time, we limited the number of neighbouring grids that T-Share explores to find a match to 80, that translates to approximately 4km. Thus, in our T-Share setting, no car will take a detour of more than 4km to serve a request, which can be considered a reasonable maximum detour for a car in NY city. As can be seen from Figure 4a, XAR takes significantly less time to search rides than T-Share. Even in the worst case, which appears after 95th percentile of the requests, (Figure 4a), the longest that XAR took to search a ride is about 3ms, whereas T-Share required almost 1 second to come up with all possible matches. Figure 4b and Figure 4c show the comparison of time taken by each system to create and book rides, respectively. Having shown that XAR searches substantially more efficiently than T-Share, we were interested in knowing if the search optimization affected the time taken to create and book rides as we update indexes during these operations (section VI). We see that T-Share certainly creates and books rides faster than XAR. However, it is evident that the orders of time taken by T-Share and XAR for these operations are quite close and comparable.

Since T-Share needs to calculate shortest path for searching rides, we called OpenTripPlanner to find shortest distances between locations for the purpose. We suspected that shortest path calculation resulted in longer search times for T-Share. Hence, in an alternate setting, we removed calls to trip planner and instead computed distances using haversine formula, which takes negligible constant time. We also varied the number of potential matches to be returned, k, in this experiment to evaluate the performance of the systems for scenarios where we need more than one match to be returned for a request (refer I). We ran simulations in this setting for $k = 1, 2, \ldots, 25$. It is evident from Figure 5a that the search time taken by T-Share increases linearly with an increase in k, despite removing the shortest path calculation. On the other hand, the value of k has little or almost no effect on the time required by XAR to search rides, it continues to search rides within 0.5ms on average for all the values of k. Thus, XAR emerges as a clear winner. Notably, XAR is able to search faster and can serve more requests than T-Share in a given time span, though it does not have a limitation on car detour allowed in matching, unlike T-Share. This experiment proves that higher search time of T-Share is not just because of shortest path calculation, but also due to the way rides are indexed.

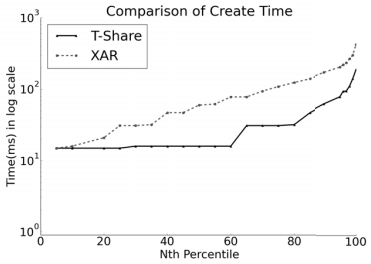
Next, it was interesting to observe how the two systems responded to high look-to-book ratio resulting from integration with an MMTP which combines ride sharing with other modes of transport (section IX). We compare the performance of the two systems for look-to-book ratio, $r = 1, 5, 10, \ldots, 1000$. The time taken by each system to book a ride after performing r searches is plotted in 5b, on logarithmic scale. While T-Share takes less time than XAR for $r = 1$, it performs increasingly poorly for $r > 1$ (which is a realistic case). As Figure 5b shows, with increase in r, time taken by T-Share increases in an order much higher than that for XAR. For $r = 1000$, T-Share took about 42 seconds, whereas XAR took slightly longer than 1 second.

Thus, all the experiments in this set show that XAR significantly outperforms the state-of-the-art in searching rides. T-Share is certainly faster in creating and booking rides. However, in real-world ride sharing systems, optimizing search operation becomes more important as such systems have high look-to-book ratio and there are more calls to search than creation or booking of rides. The case of high look-to-book ratio also arises when an MMTP incorporates ride sharing as one of its modes (section IX). One might wonder why we assume a high look to book ratio. To put light on that, we took the actual data from the Go-LA app [27], which is an MMTP for Los Angeles on which we are beta-testing our application. From this data, we found that on an average there are 8 trip plans for each request made. Further, each trip-plan has on an

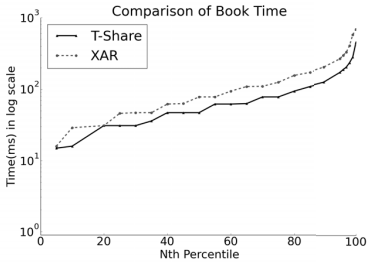
[5]The original implementation of T-Share is not publicly available. Hence, we implemented T-Share to resemble the description in [6].



(a) Total time taken by XAR and T-Share to search all possible matches

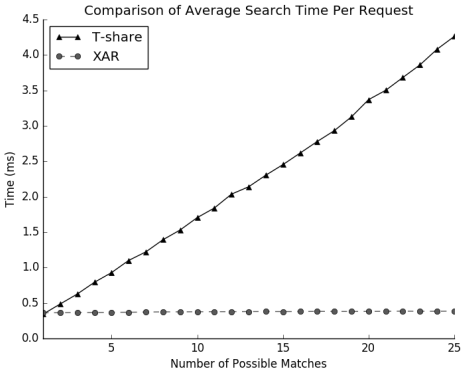


(b) Total time taken by XAR and T-Share to create a ride

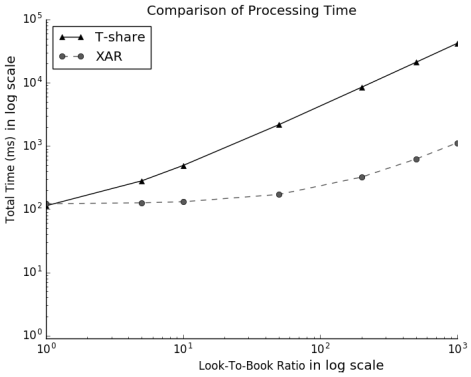


(c) Total time taken by XAR and T-Share to book a ride

Fig. 4: Comparison of average case time taken by XAR and T-Share to search, create and book rides



(a) Average time taken by XAR and T-Share to search k possible matches



(b) Total time taken by XAR and T-Share to process queries for different look-to-book ratios

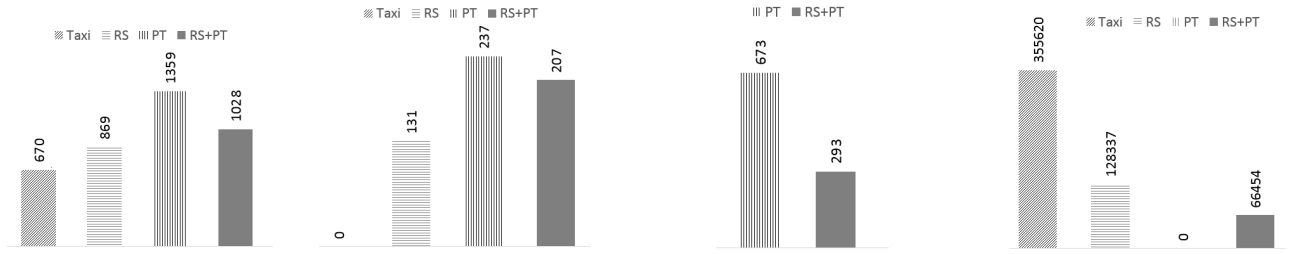
Fig. 5: Comparison of time taken by XAR and T-Share to search in different real-world scenarios

average 4 legs (i.e. 3 hops). As we explained for the Enhancer mode, this would mean $8 * \binom{3+1}{2} = 48$ ride-sharing searches being made for each request. If we assume that 1 in every 10 persons opts for ride-sharing, the look-to-book ratio becomes as high as $10 * 48 = 480$. And this is just the average case. So, we might assume a very high look-to-book ratio in pragmatic scenarios.

3) Combining XAR with Other Modes of Transport: In this experiment, we compare four modes of transportation, viz., taxi, public transport, ride sharing, and public transport combined with ride sharing. We compare the quality of travel across different modes for following metrics: end-to-end travel time, walking time and waiting time. We also evaluate the usefulness of combining ride sharing with other transport modes in terms of number of cars needed to serve the requests, thus helping in creating an environment-friendly transport option and reducing traffic on the road.

All these metrics can be obtained trivially from the data set when the mode of transport is taxi. We ran experiments for the other three modes of transport. In each case, we used the same requests as the taxi ride request but considered the scenario where these requests are served by other modes. For public transport mode, we obtained GTFS data for public transit modes in NY city from [28] and cleaned it. Then we tried to serve all the requests with public transport using OpenTripPlanner as the MMTP and measured the aforementioned metrics. For the shared ride mode, we assumed all the requesters to be willing to ride share and created ride sharing matches using XAR and measured the metrics. For public transport combined with ride sharing, we tuned OpenTripPlanner and XAR to work in aider mode as explained in the Section IX. For this experiment, we consider segments with walking distance exceeding 1 km or waiting time exceeding 10 mins for a single segment to be infeasible. The results are plotted in Figure 6.

As expected and is obvious, taxi outperforms other modes in terms of travel time, waiting time and walking distance, but is not environment friendly and in fact, worsens the road traffic as it needs more cars on the road. Public transport, which is environment-friendly in that it carries more people, thus resulting in less emission and reduction in the number of vehicles on the road, performs poorly in other metrics. Ride sharing system has a right trade-off between quality of travel and environment friendliness. It reduces almost 64% of car usage with a 30% increase in travel time. Interestingly, ride sharing combined with public transport has improved the quality of travel than standalone public transport and is more environment friendly than ride sharing. As we can see total walking time is reduced by 56% and travel time is reduced by 30% as compared to public transport and number of cars needed is reduced by almost 50% as compared to standalone ride sharing.



(a) Avg. Travel time (in seconds) (b) Avg. Waiting time (in seconds) (c) Avg. Walking time (in seconds) (d) Number of cars used

Fig. 6: Comparing different modes of transportation Taxi, Ride Sharing (RS), Public Transport (PT) and Ride sharing with public transport (RS+PT)

C. Summary

It is evident from the experimental evaluations that, XAR performs significantly better compared to the state-of-the-art (T-Share, in this case). Although T-Share performed relatively better (although by a small margin) for ride create and book operations, XAR outperforms it in the ride search operation thus verifying it to be a search-optimized system. This is crucial because integrating such systems with MMTP will have high look-to-book ratio (i.e., the number searches are far greater than the number of bookings). Thus, XAR establishes itself as a highly scalable dynamic ride sharing system, owing to its optimized search operation. XAR also proves to be suitable in providing an environment-friendly transport option when combined with public transport using the proposed modes of integration as it results in fewer vehicles on road as demonstrated by our evaluations.

XI. Conclusions

We described a search optimized, accurate within bounded error tolerances, scalable dynamic ride sharing system XAR. The system allows peer-to-peer ride sharing as stand-alone or integrated with an MMTP. The design and analysis of the algorithms and the data structures is novel and theoretically validated, as well as extensive empirical evaluations have been done using real data that show the effectiveness of such a system.

References

[1] "Carpooling initiative by bangalore traffic police," http://www.medianama.com/2015/09/223-carpooling-bangalore-police/, last access: 29-Feb-2016.

[2] "Ridesharing (by us department of energy)," http://www.afdc.energy.gov/conserve/ridesharing.html, last access: 29-Feb-2016.

[3] "Uberpool," http://blog.uber.com/uberpool, last access: 29-Feb-2016.

[4] "Lyftline," https://www.lyft.com/line, last access: 29-Feb-2016.

[5] "zimride," https://zimride.com/, last access: 2-Mar-2016.

[6] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service," in *29th IEEE International Conference on Data Engineering (ICDE)*, 2013, pp. 410–421.

[7] "Multi-modal trip planner," https://en.wikipedia.org/wiki/Intermodal_journey_planner, last access: 29-Feb-2016.

[8] "NYC taxi trips," http://www.andresmh.com/nyctaxitrips/, last access: 23-March-2016.

[9] M. Furuhata, M. Dessouky, F. Ordóñez, M.-E. Brunet, X. Wang, and S. Koenig, "Ridesharing: The state-of-the-art and future directions," *Transportation Research Part B: Methodological*, vol. 57, pp. 28–46, 2013.

[10] Y. Huang, F. Bastani, R. Jin, and X. S. Wang, "Large scale real-time ridesharing with service guarantee on road networks," *Proceedings of the VLDB Endowment*, vol. 7, no. 14, pp. 2017–2028, 2014.

[11] B. Cao, L. Alarabi, M. F. Mokbel, and A. Basalamah, "Sharek: A scalable dynamic ride sharing system," in *16th IEEE International Conference on Mobile Data Management (MDM)*, vol. 1, 2015, pp. 4–13.

[12] R. Geisberger, D. Luxen, S. Neubauer, P. Sanders, and L. Volker, "Fast detour computation for ride sharing," *arXiv preprint arXiv:0907.5269*, 2009.

[13] D. Pelzer, J. Xiao, D. Zehe, M. H. Lees, A. C. Knoll, and H. Aydt, "A partition-based match making algorithm for dynamic ridesharing," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 5, pp. 2587–2598, 2015.

[14] N. A. Agatz, A. L. Erera, M. W. Savelsbergh, and X. Wang, "Dynamic ride-sharing: A simulation study in metro atlanta," *Transportation Research Part B: Methodological*, vol. 45, no. 9, pp. 1450–1464, 2011.

[15] M. Ota, H. Vo, C. Silva, and J. Freire, "A scalable approach for data-driven taxi ride-sharing simulation," in *IEEE International Conference on Big Data (Big Data)*, 2015, pp. 888–897.

[16] D. O. Santos and E. C. Xavier, "Dynamic taxi and ridesharing: A framework and heuristics for the optimization problem," in *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI)*, vol. 13, 2013, pp. 2885–2891.

[17] N. Agatz, A. Erera, M. Savelsbergh, and X. Wang, "Optimization for dynamic ride-sharing: A review," *European Journal of Operational Research*, vol. 223, no. 2, pp. 295–303, 2012.

[18] C.-F. Liaw, C. C. White, and J. Bander, "A decision support system for the bimodal dial-a-ride problem," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 26, no. 5, pp. 552–565, 1996.

[19] K.-T. Lee, D.-J. Lin, and P.-J. Wu, "Planning and design of a taxipooling dispatching system," *Transportation Research Record: Journal of the Transportation Research Board*, no. 1903, pp. 86–95, 2005.

[20] X. Li and L. Quadrifoglio, "Feeder transit services: choosing between fixed and demand responsive policy," *Transportation Research Part C: Emerging Technologies*, vol. 18, no. 5, pp. 770–780, 2010.

[21] "OpenStreetMap," https://www.openstreetmap.org.

[22] "OpenTripPlanner," http://www.opentripplanner.org.

[23] U. Feige, "A threshold of ln n for approximating set cover (preliminary version)," in *Proceedings of the twenty-eighth annual ACM Symposium on Theory of Computing (STOC)*, 1996, pp. 314–318.

[24] T. F. Gonzalez, "Clustering to minimize the maximum intercluster distance," *Theoretical Computer Science*, vol. 38, pp. 293–306, 1985.

[25] "Classmexer," http://www.javamex.com/classmexer, last access: 23-March-2016.

[26] "Google places api," https://developers.google.com/places, last access: 23-March-2016.

[27] "Go-LA," http://golaapp.com/.

[28] "Metropolitan transportation authority," http://www.mta.info/, last access: 23-March-2016.