ZAC: A zone path construction approach for effective real-time ridesharing

ZAC: A Zone pAth Construction Approach for Effective Real-Time Ridesharing

Meghna Lowalekar, Pradeep Varakantham, Patrick Jaillet[†]
School of Information Systems, Singapore Management University
[†]Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, USA
meghnal.2015@phdis.smu.edu.sg, pradeepv@smu.edu.sg, jaillet@mit.edu

Abstract

Real-time ridesharing systems such as UberPool, Lyft Line, GrabShare have become hugely popular as they reduce the costs for customers, improve per trip revenue for drivers and reduce traffic on the roads by grouping customers with similar itineraries. The key challenge in these systems is to group the right requests to travel in available vehicles in real-time, so that the objective (e.g., requests served, revenue or delay) is optimized. The most relevant existing work has focussed on generating as many relevant feasible (with respect to available delay for customers) combinations of requests (referred to as trips) as possible in real-time. Since the number of trips increases exponentially with the increase in vehicle capacity and number of requests, unfortunately, such an approach has to employ ad hoc heuristics to identify relevant trips.

To that end, we propose an approach that generates many zone (abstraction of individual locations) paths – where each zone path can represent multiple trips (combinations of requests) – and assigns available vehicles to these zone paths to optimize the objective. The key advantage of our approach is that these zone paths are generated using a combination of offline and online methods, consequently allowing for the generation of many more relevant combinations in real-time than competing approaches. We demonstrate that our approach outperforms (with respect to both objective and runtime) the current best approach for ridesharing on both real world and synthetic datasets.

1 Introduction

Real-time taxi sharing platforms, such as UberPool, Lyft Line and GrabShare, etc. and on demand shuttle services such as Shotl, Beeline and GrabShuttle, etc. have become hugely popular in recent years due to significant benefits provided by them. These ridesharing and shuttle services reduce costs for the customer and improve per trip revenue for drivers. In addition, they also help in reducing the traffic congestion as they allow customers with similar itineraries to share a vehicle. Other shared mobility services, such as car sharing, courier services, scooter sharing, bikesharing, etc. also have a similar underlying problem and the approach (with minor extensions) presented in this paper can be used for those problems.

The ridesharing problem (Alonso-Mora et al. 2017; Bei and Zhang 2018) is related to the vehicle routing (Ritzinger, Puchinger, and Hartl 2016) and multi-vehicle pickup and delivery problems (Parragh, Doerner, and Hartl 2008; Yang, Jaillet, and Mahmassani 2004), where customer demand should be picked up from their origin locations and dropped at their destination locations while satisfying vehicle capacity and delay constraints. Earlier work on these problems has focussed on traditional integer programming approaches which are limited to small scale problems of 8 vehicles and 96 requests (Ropke, Cordeau, and Laporte 2007; Ropke and Cordeau 2009). It is also possible to model the ridesharing problem as a Markov Decision Process (MDP) (Puterman 2014) because the underlying problem is a multi-step matching problem of demand and supply. But as the number of states and actions are exponential in the number of agents (vehicles) and the number of vehicles is in the thousands, it is even difficult to specify the model.

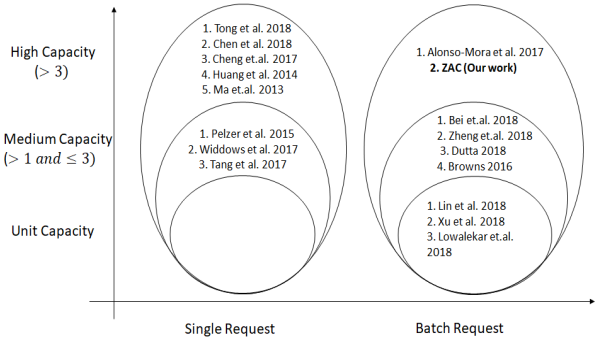


Figure 1: Related Work

In recent times, many heuristic approaches (Ma, Zheng, and Wolfson 2013; Agatz et al. 2011) have been proposed to solve the real-time taxi ridesharing problem. Most existing research can be categorized as either:

- Finding a greedy solution – that assigns one request at a time to the best available vehicle – for high capacity vehicles (shuttles, buses etc.) (Tong et al. 2018; Huang et al. 2014; Ma, Zheng, and Wolfson 2013) **or** on
- Finding a batch solution – that assigns all active requests together in a batch to the available vehicles – for

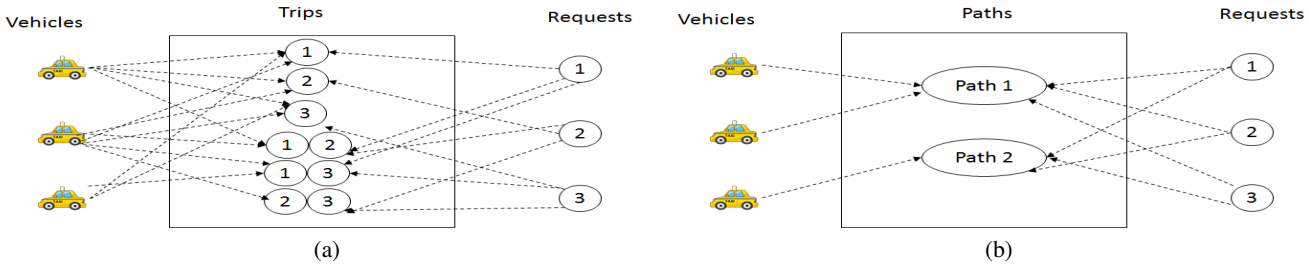


Figure 2: (a) Representation of RTV graph generated by the model in (Alonso-Mora et al. 2017) for capacity 2. (b) Representation of RPV graph generated by ZAC approach.

low capacity vehicles (e.g., taxis) (Bei and Zhang 2018; Dutta 2018; Zheng, Chen, and Ye 2018).

Related work is succinctly summarized in Figure 1. The greedy solution is faster to compute but the quality of solution obtained is typically poor. On the other hand, batch solution takes significantly more time to compute but the solution quality is significantly better than greedy solution.

Similar to Alonso et al. (Alonso-Mora et al. 2017), we focus on the most challenging category of obtaining a batch solution for high capacity vehicles in real-time. The approach by Alonso et.al. (Alonso-Mora et al. 2017) is a generalization of the greedy approach typically employed by taxi companies (Widdows et al. 2017; Tang et al. 2017; Browns 2016) and is divided into two parts.

- The first part constructs a RTV (Request Trip Vehicle) graph. A trip in an RTV graph corresponds to a combination of requests that is feasible (with respect to available delay for customers). There is an edge between request and trip if the request is a part of the trip and there is an edge between trip and vehicle if the vehicle can serve all the requests in that trip.
- From all allowable allocations of vehicles to trips, the second part computes an optimal allocation of vehicles to trips that minimizes delay or maximizes the number of requests served.

This approach is limited in scalability as the set of possible trips increases exponentially with the increase in the number of requests and capacity of vehicles. To ensure scalability and address the key challenge of identifying as many relevant trips as possible in real-time, this approach employs ad hoc heuristics (e.g., limiting time available and edges in RTV graph).

We propose a new approach called ZAC (**Z**one p**A**th **C**onstruction), that employs two crucial ideas to identify significantly more relevant trips in real-time:

- ***Focus on zone paths instead of trips:*** A zone path is a path that connects zones (a zone is an abstraction for multiple individual locations) and therefore it can group multiple trips that have "nearby" or "on the way" pickups and drop-offfs. This focus on zone paths helps automatically capture multiple *relevant* trips with one zone path.
- ***Offline-online computation of zone paths:*** Since, we focus on zone paths, we can generate partial zone paths offline. This helps capture more number of relevant trips online in real-time, where the partial paths are completed.

Instead of an RTV (Request Trip Vehicle) graph in Alonso *et al.*'s (Alonso-Mora et al. 2017) approach, we construct an RPV (Request Path Vehicle) graph, where we associate requests and vehicles to zone paths. This is shown in Figure 2. Once the RPV is constructed, we then employ a scalable integer linear program to find the optimal assignment (e.g., maximize revenue, maximize the number of requests served or minimize the delay) of vehicles and requests to paths.

We compare our approach against the Alonso *et al.*'s (Alonso-Mora et al. 2017) approach, referred to as TBF, on synthetic and two real world datasets and show that our approach not only is very efficient (with respect to runtime) but also serves more requests. On real datasets, as shown in the experimental results, ZAC obtains upto 4% gain over TBF. This is a significant gain, compared to the 0.5% gain achieved by a real taxi company (Xu et al. 2018).

There is also a recent increase in the popularity of on demand shuttle services (Shotl 2018; Beeline 2016; Grab 2018). These shuttles have fixed pickup/drop-off points which is a very small subset of complete road network of a city. Therefore, we also perform experiments on a synthetic dataset introduced by Bertsimas *et al.* (Bertsimas, Jaillet, and Martin 2018), where the first mile and last mile transportation requests are simulated. In these cases, ZAC obtains a staggering 20% gain over TBF, further supporting our claim that ZAC is suitable for real-time ridesharing with higher capacities.

2 Rideshare Matching Problem (RMP)

Real-time ridesharing is a service provided by platforms such as Uber, Lyft etc. for arranging shared rides for multiple customers at a very short notice. Customers request for a shared ride from a source to destination. The platform then groups all those requests that can share a ride – based on whether the delay[1] of reaching the destination is less than a given threshold for all requests sharing the ride. These services are popular because customers are ready to accept a delay in exchange for a reduced fare. However, after an acceptable threshold, delay cannot be compensated with mon-

[1]Time taken to reach the destination using the shared ride minus the time taken using an individual ride

etary benefits. Therefore, when making groups of customer requests, platforms need to ensure that matching algorithms find groupings which do not increase the delay of individual customers beyond an acceptable threshold[2].

Formally, we define RMP using the following tuple:

$$< \mathcal{G}, \mathcal{D}, \mathcal{V}, \mathcal{C}, \tau, \lambda, \Delta >$$

- $\mathcal{G} = (\mathcal{L}, \mathcal{E})$ is a graph with the vertices as the set of locations. For e.g., as considered in previous works (Alonso-Mora et al. 2017) the graph $\mathcal{G}$ is the road network with the set $\mathcal{L}$ including all street intersections in the road network of a city and $\mathcal{E}$ denotes the set of road segments. Travel Time ($\mathcal{T}$) and shortest paths ($\mathcal{S}_p$) between all location pairs in set $\mathcal{L}$ is pre-computed and stored.
- $\mathcal{D}$ denotes the set of customer requests. Each element $j \in \mathcal{D}$ is represented using the tuple: $\langle o_j, d_j, a_j \rangle$, where $o_j, d_j \in \mathcal{L}$ denote the origin and destination location and a_j denotes the arrival time of the request j.
- $\mathcal{V}$ denotes the set of vehicles. Each element $i \in \mathcal{V}$ is represented using the tuple: $\langle \nu_i, \omega_i, \mu_i, q_i \rangle$. $\nu_i \in \mathcal{L}$ denotes the initial location of vehicle i, ω_i denotes the time at which vehicle first becomes available at ν_i, μ_i denotes the capacity of vehicle i and q_i denotes the set of customer requests assigned to vehicle i. Each element j of q_i is represented using the tuple: $\langle o_j, d_j, a_j \rangle$, where o_j, d_j, a_j are as described in the demand tuple above.
- $\mathcal{C}$ represents the objective (e.g. revenue, number of requests served, etc.), with $\mathcal{C}_{ij}^t$ denoting the value obtained on assigning request j to vehicle i at decision epoch t.
- τ denotes maximum allowed wait time for a request (in seconds). The wait time is defined as the difference between the arrival time of a requests and the time at which vehicle picks the customer from its origin.
- λ denotes maximum allowed travel delay for requests (in seconds). The travel delay is the total delay experienced by the customer to reach its destination location [3].
- Δ denotes the decision epoch duration in seconds, i.e., the algorithm is executed every Δ seconds.

The goal in RMP is to assign the incoming customer requests to the vehicles such that the capacity constraints, maximum wait time and delay constraints are satisfied.

3 ZAC: A Zone pAth Construction Approach for solving RMP

Given the importance of zone path to ZAC, we first define and explain about zone and zone path. We then describe the intuitive advantages of using zone paths and then we explain the ZAC algorithm.

Definition 1 ***Zone****: refers to an abstracted location obtained by clustering locations in set $\mathcal{L}$.*

[2]Threshold values can be potentially learnt by surveying different customers.

[3]If t_j^d denotes the time at which a request j is dropped at its destination then the travel delay is given by $t_j^d - (a_j + \mathcal{T}(o_j, d_j))$.

In this work, we investigated Grid Based Clustering (GBC), Hierarchical Agglomerative Clustering with Complete Linkage (HAC_MAX) and Hierarchical Agglomerative Clustering with Mean Linkage (HAC_AVG) to cluster locations into zones. We use these methods as they do not require prior knowledge about the number of clusters and have been used in earlier works on similar problems (Ma, Zheng, and Wolfson 2013; Hasan et al. 2018).

Definition 2 ***Zone path****: refers to an ordered sequence of nodes, where each node corresponds to either a location from set $\mathcal{L}$ or a zone.*

There are two key advantages to a zone path:

- Zone path represents multiple trips that have "nearby" or "on the way" pickups and drop-offs; and
- Zone path can be generated at different levels of granularity (e.g., individual locations, communities) depending on the time available.

Due to these two advantages, zone paths assist in identifying more relevant trips (combinations of requests) within a given amount of runtime. We further enhance the ability to identify more relevant trips within limited runtime, by generating zone paths partially offline and completing them in real-time depending on the set of active requests.

We generate the zone path of time span τ offline and complete the rest of the zone path online. This is because requests can be picked up only in initial τ seconds [4]. Therefore, partial zone paths generated offline automatically provide a pickup order for the active requests. As a result, online, we only need to compute drop-off order while ensuring that the delay constraints are not violated. This is in contrast to Alonso *et al.*'s (Alonso-Mora et al. 2017) approach, where both pickup and drop-off order along with the delay feasibility have to be computed online.

Intuitively, the inherent nature of zone paths to capture multiple relevant trips coupled with the extra time made available online due to offline computation of partial zone paths enables ZAC to consider significantly more relevant trips in real-time.

Due to abstraction of locations into zones, the travel time is approximately represented when considering zone paths. This can result in longer wait times or longer estimate of wait times than a path over locations in set $\mathcal{L}$. Customers prefer to have a shorter wait time pre-process (Dube-Rioux, Schmitt, and Leclerc 1989; Maister and others 1984), i.e., before pick-up in this case. Therefore, it is essential to reduce this approximation in travel time computation during pickup. We reduce this approximation by generating offline partial paths at the level of locations. This is another benefit of having an offline partial path.

ZAC is an offline-online approach for solving the RMP every few seconds on active requests and available vehicles by using offline generated partial paths. The key components of the ZAC algorithm are as follows:

[4]τ is typically 300 and we experiment with values between 120-420 seconds.

- Offline: generation of all partial location paths of time span, τ from every location.
- Online: generation of RPV graph by loading and processing offline partial paths, completing the partial paths and identifying edges in RPV graph.
- Online: finding optimal assignment of requests to paths to vehicles by using an efficient integer (0/1) linear optimization

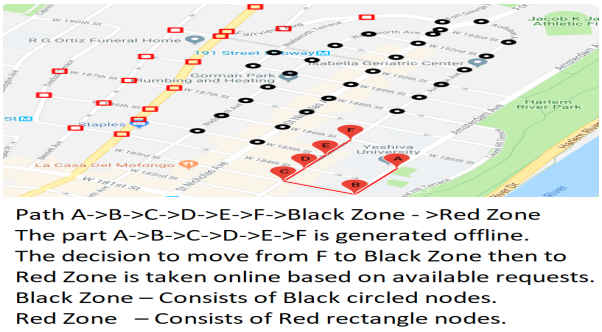


Figure 3: Example Zone Based Path.

Example 1 *Figure 3 provides an example of a zone path generated using ZAC. There is a partial zone path (generated offline) over individual locations (i.e., $A \rightarrow \ldots \rightarrow F$) and the completion of that zone path (online) using larger zones (black and red).*

3.1 Offline: Partial Paths Generation

The main challenge with generating a partial path at the level of individual locations – even for a time span of only maximum wait time, τ – is the time taken to generate all the paths. Therefore, we compute these partial paths (of span τ seconds) offline by generating all simple paths of duration τ in the network $\mathcal{G}$. The number of all possible paths grows exponentially with the increase in the value of τ and increase in the number of locations. In case all possible paths can not be generated due to memory constraints, we can employ a data driven approach (based on historical data) to generate paths which have high likelihood of grouping large number of requests. These offline partial paths ($\mathcal{P}_{off}$) are stored by indexing on the start location and start time ($\mathcal{P}_{off}[l,t]$) [5]. The start time associated with the path indicates the time at which the first node (location) in the path is visited. For a clear explanation, two paths starting at the same location but having different start times are considered different. This is because vehicles can become available at the same location but at different time. These offline partial paths are further indexed by the location and time of each node present in the path for quick online processing.

3.2 Online

We now describe the crucial online component of ZAC that generates the RPV graph and finds the optimal match on the generated RPV graph. The pseudocode for the online component ZAC-Online is provided in Algorithm 1. After loading the offline computed partial paths, travel times and shortest paths, at every decision epoch, ZAC-Online considers the currently available batch of requests and current vehicle status to find the optimal assignment in two steps: (1) Generation of the Request, Path and Vehicle (RPV) graph and (2) Finding optimal match in RPV graph using a linear integer optimization model.

Algorithm 1 ZAC-Online()

1: $t = starttime$ (in seconds)
2: $\mathcal{P}_{off} = \bigcup_{\substack{l \in \mathcal{L}, \\ t' < \tau}} P_{off}[l,t'] =$ LoadOfflinePartialPaths()
3: $\mathcal{T} =$LoadTravelTimes(), $\mathcal{S}_p =$LoadShortestPaths()
4: **while** $t < endtime$ **do**
5: $t_1 = t - starttime$
6: **if** $(t_1)\%\Delta == 0$ **then**
7: $\mathcal{D}, \mathcal{V} \leftarrow$GetCurrentDemand-VehicleStatus(t)
8: $\mathcal{P}, Pv, Pr, b, N =$ GenerateRPVGraph($t, \mathcal{P}_{off}, \mathcal{D}, \mathcal{V}, \mathcal{T}, \mathcal{S}_p$)
9: SolveOptimization($\mathcal{P}, Pv, Pr, b, N$)
10: $t = t + 1$

We now describe the two steps of ZAC in detail.

Generation of the RPV graph As shown in Algorithm 2, there are three key steps to RPV graph generation: (1) Online processing of Offline Partial Paths; (2) Online Partial Zone Path Completion; (3) Identifying edges in the RPV graph.

Algorithm 2 GenerateRPVGraph($t, \mathcal{P}_{off}, \mathcal{D}, \mathcal{V}, \mathcal{T}, \mathcal{S}_p$)

1: $\mathcal{P}'_{off}, \mathcal{R}' =$ ProcessOfflinePartialPaths($t, \mathcal{P}_{off}, \mathcal{D}, \mathcal{V}, \mathcal{T}, \mathcal{S}_p$)
2: $\mathcal{P}, \mathcal{R}'' =$ OnlineCompletion($t, \mathcal{P}'_{off}, \mathcal{R}', \mathcal{T}, \mathcal{S}_p$)
3: $\mathcal{P}, Pv, Pr, b, N =$ IdentifyEdgesRPVGraph($t, \mathcal{P}, \mathcal{D}, \mathcal{V}, \mathcal{R}''$)
4: **return** $\mathcal{P}, Pv, Pr, b, N$

Online Processing of Offline Partial Paths: The offline generated partial paths are processed online based on the current available demand and vehicle status (vehicle location, currently assigned requests to vehicle) as shown in Algorithm 3. Steps 1-2 ensure that we consider only those paths which start at a location and time where atleast one vehicle is present and these paths are processed in parallel using multiple threads. The GetPathsFromIndex function returns the set of offline partial paths which visit the given location within given time interval and uses the pre-computed offline indexes for quick online retrieval. Step 12 stores the set of destination locations of the currently available requests grouped along the path (based on the pickup). In addition to the destination location, we also store the lower and upper bound on the time by which the location should be visited. Similarly, in step 19, we store the destination locations of the requests previously assigned to vehicles. In, step 19, we consider only those paths which can potentially satisfy all the previously assigned requests for a vehicle. This is because a vehicle will be assigned to a

[5] We discretize the time at the level of 10 seconds.

path if and only if it can serve all the previously assigned requests. In addition, a vehicle should deviate from its current path only if it can be assigned to a new request, therefore, we consider only those paths which can pick at least one of the newly available request.

Algorithm 3 ProcessOfflinePartialPaths(t,$\mathcal{P}_{off}$,$\mathcal{D}$,$\mathcal{V}$, $\mathcal{T}$,$\mathcal{S}_p$)

1: $\mathcal{P}'_{off} = [], Lt_v = \bigcup_{i \in \mathcal{V}} (\nu_i, \omega_i)$
2: Create H threads. Each thread h processes $\mathcal{P}^h_{off} = \bigcup_{k' \in Lt^h_v} \mathcal{P}_{off}[k']$, ,s.t., $Lt^a_v \cap Lt^b_v = \phi, \forall a \neq b$ and $\cup_h Lt^h_v = Lt_v$
3: **for** each thread h **do**
4: $\quad \mathcal{V}' \subset \mathcal{V}, s.t., \forall i' \in \mathcal{V}', (\nu_{i'}, \omega_{i'}) \in Lt^h_v$
5: $\quad$ **for** $j \in \mathcal{D}$ **do**
6: $\quad\quad \mathcal{P}^{h,j}_{off}$ = GetPathsFromIndex($\mathcal{P}^h_{off}, o_j, a_j - t, a_j - t + \tau$)
7: $\quad\quad$ **for** each path $k \in \mathcal{P}^{h,j}_{off}$ **do**
8: $\quad\quad\quad lb_j = a_j - t + \mathcal{T}(o_j, d_j), ub_j = lb_j + \lambda$
9: $\quad\quad\quad$ **if** $\mathcal{R}[k]$ contains d_j **then**
10: $\quad\quad\quad\quad \mathcal{R}[k][d_j][1] = max(R[k][d_j][1], ub_j)$
11: $\quad\quad\quad$ **else**
12: $\quad\quad\quad\quad \mathcal{R}[k].add(d_j, (lb_j, ub_j))$
13: $\quad\quad\quad \mathcal{R}_p[k].add(o_j)$
14: $\quad\quad\quad$ **if** $lb_j < \tau$ and k visits d_j **then**
15: $\quad\quad\quad\quad \mathcal{R}_p[k].add(d_j)$
16: $\quad\quad\quad$ **else if** $ub_j < \tau$ and k does not visit d_j **then**
17: $\quad\quad\quad\quad \mathcal{R}[k].remove(d_j, (lb_j, ub_j))$
18: $\quad$ **for** $i \in \mathcal{V}'$ **do**
19: $\quad\quad \mathcal{R}[k], \mathcal{R}_p[k]$ = GetPathsForVehicle($i, q_i, \mathcal{R}[k], \mathcal{R}_p[k], \mathcal{P}_{off}$)
20: $\quad$ **for** each path k **do**
21: $\quad\quad$ **if** $|\mathcal{R}_p[k]| > 0$ **then**
22: $\quad\quad\quad$ Remove nodes not in $R_p[k]$, update $\mathcal{R}_k$ using $\mathcal{T}, \mathcal{S}_p$
23: $\quad\quad\quad \mathcal{P}'^h_{off}.add(k)$
24: **for** each thread h **do**
25: $\quad \mathcal{P}'_{off}.addAll(\mathcal{P}'^h_{off})$
26: **return** $\mathcal{P}'_{off}, \mathcal{R}$

Steps 14 and 19 ensure that if the drop-off location of request can be visited in the partial path, then it is considered in the processing. In the end, in steps 20-22, as an optimization, we only keep those locations in the partial paths which correspond to a pickup or drop-off location and update the travel time and path between the locations using $\mathcal{T}$ and $\mathcal{S}_p$.

The offline generated partial paths significantly improve the scalability of completing the path online using exhaustive search. This provides more time online for considering more zone paths and hence more relevant trips.

Online Partial Zone Path Completion: The partial paths generated offline are completed online using exhaustive search starting at the end location of the partial path as shown in the Algorithm 4. By online processing of offline paths (refer Algorithm 3), we can identify the requests that can be associated with each of the offline generated partial paths (based on their pickup location). We use the destination locations of these requests to complete the remaining

Algorithm 4 OnlineCompletion($t, \mathcal{P}'_{off}, \mathcal{R}', \mathcal{T}, \mathcal{S}_p$)

1: $\mathcal{P} = [], \mathcal{R} = []$
2: Create H threads.
Each thread h processes $\mathcal{P}'^h_{off} \subset \mathcal{P}'_{off}$,s.t., $\mathcal{P}'^h_{off} \cap \mathcal{P}'^{h'}_{off} = \phi, \forall h \neq h'$ and $\cup_h \mathcal{P}'^h_{off} = \mathcal{P}'_{off}$
3: **for** each thread h **do**
4: $\quad \mathcal{P}^h_{on} = [], \mathcal{R}^h_{on} = []$
5: $\quad$ **for** each path k **do**
6: $\quad\quad z =$ getAppropriateZoneSize($\mathcal{R}[k], M$)
7: $\quad\quad \mathcal{R}' =$ convert($\mathcal{R}[k], z$)
8: $\quad\quad \mathcal{P}^h_{on}, \mathcal{R}^h_{on}$ = ExhaustiveSearch($end\_node(k), \mathcal{R}', \mathcal{P}^h_{on}, \mathcal{R}^h_{on}$)
9: **for** each thread h **do**
10: $\quad \mathcal{P}.addAll(\mathcal{P}^h_{on})$
11: $\quad \mathcal{R}.addAll(\mathcal{R}^h_{on})$
12: **return** $\mathcal{P}, \mathcal{R}$

path online. As with each destination location, we also store a lower and upper limit on the time at which it should be visited, we only explore those branches in the search tree where these time limits are satisfied. The computational complexity of online partial path completion is dependent on the number of destination locations (size of $\mathcal{R}[k]$ in Algorithm 4) and can be significant , therefore, we use zones (and not individual locations) in this step. As mentioned before, by using zones, travel time is approximately represented which can result in additional delay for requests. The additional delay introduced is dependent on the size of the zones[6] chosen. Therefore, to consider a trade-off between computational complexity and the quality of solution, we propose picking the zone sizes dynamically for each offline partial path. In order to fix the amount of dynamism in zone size, we use a parameter M that defines the number of different zone sizes that can be used in completion of offline partial paths. $M = 1$, implies static zone sizes, i.e., using zones of a fixed size for online completion of all offline partial paths. The zones of M different sizes are generated offline and in the step 6, depending on the number of destination locations and M available zone sizes, we decide the appropriate zone size for the partial path k [7].

As the partial paths are independent of each other, to further speed up the path generation process, we perform the online path completion process in parallel by creating multiple threads as shown in the pseudocode provided in Algorithm 4. Please note that the exhaustive search in step 8, will return multiple completed zone paths corresponding to a single partial path k.

For the objective of maximizing the number of requests served, the paths which start at the same location at the same time and serve a subset of requests served by another path are redundant. This is because, we check for capacity constraints in the optimization formulation presented in the next

[6]The size of the zone is defined as the time taken to travel within a zone. Zone size 0 indicates that locations in set $\mathcal{L}$ are used.

[7]In the experiments, we use $M = 4$ with zone sizes 0,60,120,300 and use the zone size which reduces the number of locations to 12 (this provides the best trade-off between runtime and solution quality and is determined based on experiments.

section. So a single path serving r requests can be used to represent all request combinations, $\sum_{i=1}^{r} \binom{r}{i}$. Therefore, the search tree in step 8 of Algorithm 4 can be pruned only to search for non redundant paths. This reduces the size of set $\mathcal{P}$ which in-turn reduces the complexity of optimization formulation presented in section 3.2.

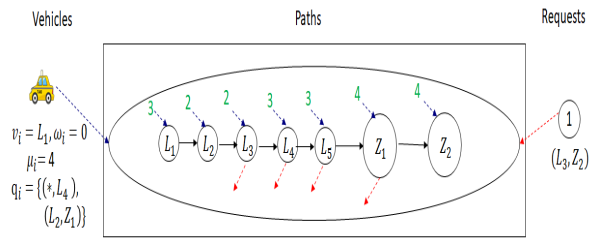


Figure 4: Representation of assignment of vehicle and request to a zone path

Identifying Edges in the RPV Graph: Once the zone paths ($\mathcal{P}$) are created using the offline-online method described above, we construct the RPV (Request Path Vehicle) graph by finding the set of requests and vehicles which can be assigned to each of the generated zone path. We use the information available from previous 2 steps about the requests and vehicles which can be assigned to these zone paths and process the paths in parallel using multiple threads to speed up the computation. This step is essential as in Algorithm 3, when same destination location has different value for the upper limit on time in step 10, we take the maximum value. Therefore, in the path generated using Algorithm 4, the delay constraint may be violated for some requests in such cases.

In this step, we ensure that a request is assigned to a zone path, if and only if, the path visits the pickup and drop-off location of a request within the delay constraints. The binary constants b (defined in Table 1 and used in optimization formulation presented next) are also populated in this step. A vehicle i represented by the tuple $(\mu_i, \nu_i, \omega_i, q_i)$ can be assigned to a zone path if the initial location of vehicle, ν_i, is same as the starting location of the path, start time of the path is same as the availability time of vehicle ω_i and currently assigned set of requests, q_i, can be served using the path. The vehicle capacity μ_i along with q_i is used to compute the number of free seats (N) in the vehicle at each zone/location.

Example 2 *Figure 4 shows a graphical view of the same for a single vehicle, request and path. The path is represented using a sequence of locations/zones in the order in which they will be visited. In Figure 4, we use blue arrows to denote the incoming flow by vehicle i assignment and green numbers indicate the number of free seats at the location/zone* [8] *for vehicle i. The red arrows indicate outgoing flow by request assignment.*

[8]Number of free seats is computed by taking μ_i and q_i of the vehicle into consideration. In figure, in the representation of q_i, we use * to indicate that customer is already present in the vehicle and provide its drop-off location.

At each location, the optimization formulation presented next, will ensure that the outgoing flow (the number of requests assigned) is less than or equal to the incoming flow (total number of free seats in the vehicles assigned to the path), i.e., at each location/zone capacity constraints are satisfied.

Finding Optimal match in RPV graph We now describe the integer linear programming optimization formulation to optimize the assignment of requests and vehicles to zone paths. $\mathcal{P}$ denotes the set of zone paths generated in previous step. $\mathcal{P}_m^n$ is used to denote the n^{th} location/zone in zone path m. Let $Pr_j \subset \mathcal{P}$ denotes the set of paths which can serve request j while satisfying delay constraints. Similarly Pv_i denotes the set of paths which can be assigned to vehicle i based on its current location ν_i, availability time ω_i and already assigned/picked-up requests q_i. Binary constants b_{jm}^n are set to 1 if the pickup location of request j is visited but drop-off location/zone is not visited along path m by n^{th} location/zone. These are computed as part of generation of RPV graph as shown in previous section. Table 1 describes the notation used in the optimization formulation.

Variable	Description
x_{jm}	Binary variable denoting if the request $j \in \mathcal{D}$ is assigned to path m.
y_{im}	Binary variable denoting if the vehicle i is assigned to the path m.
Pv_i	$Pv_i \subset \mathcal{P}$ denotes the set of paths which can be assigned to vehicle i based on its current status ν_i, ω_i and q_i.
Pr_j	$Pr_j \subset \mathcal{P}$ denotes the set of paths which can be assigned to request $j \in \mathcal{D}$.
b_{jm}^n	Binary constant: 1 if $\exists n' : n > n'\ P_m^{n'} = o_j$ $\&\& \nexists n'' : n'' < n, n'' > n'\ P_m^{n''} == d_j$
$N(i, m, n)$	Number of free seats in the vehicle i for path m at n^{th} location/zone.

Table 1: Notations

The objective of the optimization formulation described in Table 2 is to maximize the number of served requests. Constraints (2) and (3) ensure that each vehicle and each request is assigned to at most one path. Constraint (4) ensure that for every path at every location/zone capacity constraints are satisfied. The capacity constraints can be violated only while picking up a new request, therefore, the constraint (4) is redundant for the locations/zones visited after τ duration.

The formulation is run at every decision epoch, i.e., after every Δ seconds. The solution of the optimization formulation provides assignment of vehicles and requests to paths. Using these assignments, we can perform the assignment of requests to vehicles [9]. Once a vehicle is assigned to a set of requests at any decision epoch, the assignment is

[9]The paths assigned to vehicle are also updated to keep only those locations which correspond to pickup or drop-off location of assigned requests and update the travel time and path between the locations using $\mathcal{T}$ and $\mathcal{S}_p$.

SolveOptimization($\mathcal{P}, Pv, Pr, b, N$)**:**

$$\max \quad \sum_{j \in \mathcal{D}} \sum_{m \in Pr_j} x_{jm} \tag{1}$$

$$s.t. \quad \sum_{m \in Pr_j} x_{jm} \leq 1 \ ::: \ \forall j \in \mathcal{D} \tag{2}$$

$$\sum_{m \in Pv_i} y_{im} \leq 1 \ ::: \ \forall i \in \mathcal{V} \tag{3}$$

$$\sum_{j \in \mathcal{D}} x_{jm} * b^n_{jm} \leq \sum_i y_{im} * N(i,m,n) \ ::: \ \forall m \forall n \tag{4}$$

Table 2: Optimization Formulation for ZAC

not changed but the path of vehicle can change at next decision epoch to accommodate additional requests. The current set of requests assigned to a vehicle, q_i, limits the number of paths to which it can be assigned in subsequent decision epochs. The number of free seats in vehicle i for path m at location/zone n, $N(i,m,n)$ is computed based on μ_i and q_i (as shown in Figure 4) and is 0 if $m \notin Pv_i$.

Similar to Alonso *et al.* (Alonso-Mora et al. 2017), we perform a re-balancing of unassigned vehicles to high demand areas at the end of optimization formulation.

4 Experimental Setup

The goal of the experiments is to evaluate the performance of ZAC in comparison to TBF [10]. We evaluate the algorithms on following metrics: (1) Service Rate, i.e., percentage of total available requests served. (2) Runtime to compute a single step assignment. We experimented by taking demand distribution from two real world and one synthetic dataset.

The first real world dataset is the publicly available New York Yellow Taxi Dataset (NYYellowTaxi 2016), henceforth referred to as the NYDataset. The name of the other real world dataset can not be revealed due to confidentiality agreements. It is referred to as Dataset1. We use the street intersections as the set of locations $\mathcal{L}$. To find out the street intersections in real world dataset, we take the street network of the city from openstreetmap using osmnx with drive network type (Boeing 2017). From these we remove the network nodes which do not have any outgoing edges, i.e., we take the largest strongly connected component of the network. For NYDataset, as considered in earlier works (Alonso-Mora et al. 2017), we only consider the street network of Manhattan as 75% of the requests have pickup and drop-off locations in Manhattan. Moreover, less than 15% of the total requests have pickup and drop-off location in different boroughs of New York indicating that these boroughs can be solved independently.

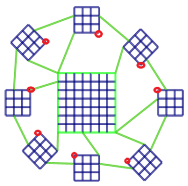
[10]The complexity of TBF increases with the increase in vehicle capacity. It is not possible to run it upto optimality. Therefore, we run it with the heuristics mentioned in the paper (0.2 second for each vehicle and keeping 30 vehicles for each request (but keeping all request edges)). We use the objective of maximizing the number of requests served for both TBF and ZAC. The objective can be changed to the objective of minimizing the delay or maximizing the revenue for both algorithms.

Dataset	**Locations** ($\lvert\mathcal{L}\rvert$)	**Edges** ($\lvert\mathcal{E}\rvert$)	**Avg No. of Requests per day**	**Avg Requests per hour (Peak)**
NYDataset	4373	9540	300237	19820
Dataset1	21212	41424	399695	23735
Synthetic	192	640	173557	8578

Table 3: Details for different datasets

Both real world datasets contain data of past customer requests for taxis at different time of the day and for different days of the week. From these datasets, we take the following fields: (1) Pickup and drop-off locations (latitude and longitude coordinates) - These locations are mapped to the nearest street intersection. (2) Pickup time - This time is converted to appropriate decision epoch based on the value of Δ. The travel time on each road segment of the street network is taken as the daily mean travel time estimate computed using the method proposed in (Santi et al. 2014).

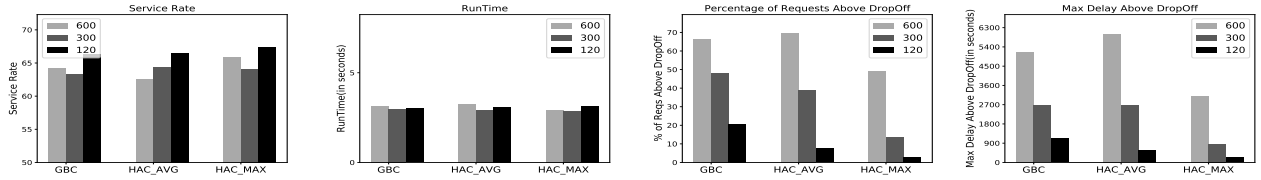
To simulate the scenario for on demand shuttle services (Shotl 2018; Beeline 2016; Grab 2018) having a small set of pickup/drop-off points in a city, we also perform experiments on a synthetic dataset introduced by Bertsimas *et al.* (Bertsimas, Jaillet, and Martin 2018). The network (Figure 5) has one downtown area represented by the big square in center and 8 suburbs. We create a train station at one node of each suburb (marked by red circle) to simulate special cases of first and last mile transportation. At each decision epoch, requests are randomly generated by taking pickup and drop-off location uniformly. In addition, every 180 seconds (frequency of arrival of train at the train stations), we generate first and last mile requests in each suburb (representing arrivals by train).



(a)

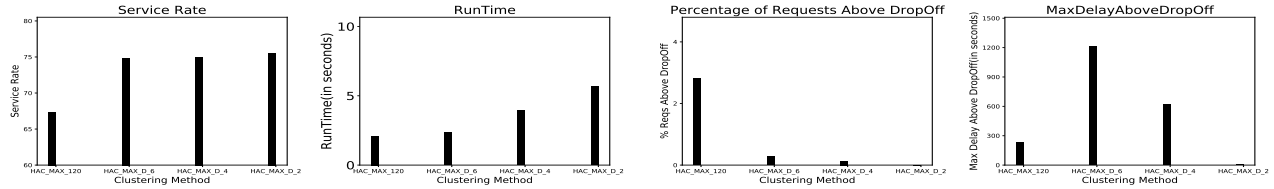
Figure 5: Street network for synthetic dataset. Train stations are marked with red.

The number of nodes/locations, edges in the street network of the city and the number of requests present in each dataset are shown in the Table 3. We evaluate the approaches for 1 hour on different days starting at different time of the day and take the average value over 5 days. We experimented with different values for each parameter as mentioned in Table 4 but due to space constraint, we show only the representative results. All experiments are run on 24 core - 2.4GHz Intel Xeon E5-2650 processor and 256GB RAM. The algorithms are implemented in Java and optimization models are solved using CPLEX 12.6.



(a) (b) (c) (d)

Figure 6: Comparison of service rate, runtime and abstraction error with different clustering methods and zone sizes for $M = 1$, number of vehicles = 1000, capacity = 10, τ =300, λ = 600 seconds



(a) (b) (c) (d)

Figure 7: Comparison of service rate, runtime and abstraction error with different values of M and zone sizes for NYDataset, number of vehicles = 1000, capacity 10, τ = 300, λ = 600 seconds

Parameter	**Values considered in Experiments**
Δ (in seconds)	10,30,60
τ (in seconds)	120,180,300,420
λ (in seconds)	240,600,900
$\|\mathcal{V}\|$	1000,2000,3000,5000,8000,10000
$\mu_i(\forall\ i)$	1,2,3,4,8,10
M	2,4,6
Clustering Method	GBC, HAC_MAX,HAC_AVG
Time of the day	08:00AM,03:00PM,06:00PM,12:00AM

Table 4: Parameter values

5 Experimental Results

In this section, we show the experimental results obtained on two real world datasets and a synthetic dataset. We first perform the experiments to identify the right clustering method and the value of M to be used for ZAC. Later, we show the comparison of ZAC and TBF on the datasets.

5.1 Identification of Right Clustering Method

We first conduct experiments by using different clustering methods, with $M = 1$, by varying the zone sizes. Zone size is taken as the intra zone travel time (in seconds). Figure 6 shows the comparison of GBC, HAC_MAX and HAC_AVG on NYDataset. We compare the service rate, runtime and abstraction error with different clustering methods and different zone sizes. We measure abstraction error by computing the percentage of requests having delay above λ and maximum delay obtained by any request which is above λ. We can observe that with HAC_MAX not only we can serve more requests but the error due to abstraction is also minimum. We also observe that as the zone size decreases, the number of requests served increases, error due to abstraction decreases with a slight increase in runtime. Based on these results, we use HAC_MAX as the clustering method for our next set of experiments.

5.2 Identification of Right Value of M

Our next set of experiments compare the service rate, runtime and abstraction error obtained using different values of M. Based on the observations made earlier, for $M = 1$, we use HAC_MAX with zone size 120. For $M > 1$, the clustering method used is HAC_MAX and we run the experiments with different values of M. We use the zone sizes as 0, 60, 120, 300, 480, 600. The zone size of 0 means that the actual locations in the street network are used. Zone size of 60 means that the intra zone travel time is 60 seconds and so on. For $M = 2$, zone sizes used are 0 and 60, for $M = 4$ zone sizes used are 0, 60, 120 and 300 and for $M = 6$, zone sizes used are 0, 60, 120, 300, 480, 600.

We show the comparison of service rate and runtime with $M = 1$ (with zone size 120) and different values of M in Figure 7. HAC_MAX_120 is used to denote that $M = 1$ with zone size 120 is used. HAC_MAX_D_$< m >$ denotes that value of M used is m. From the Figure 7 we can observe that we can serve more requests when $M > 1$, as compared to using fix large size zones. The abstraction error also reduces significantly by using $M > 1$. As the value of M is reduced, quality of solution improves with the increase in runtime. With $M = 2$ (for zone sizes 0 and 60) , the abstraction error is almost 0 but runtime also increases. With $M = 4$, the abstraction error is less than 1%.

From these experiments on NYDataset, we obtain that by clustering locations into zones using HAC_MAX and using

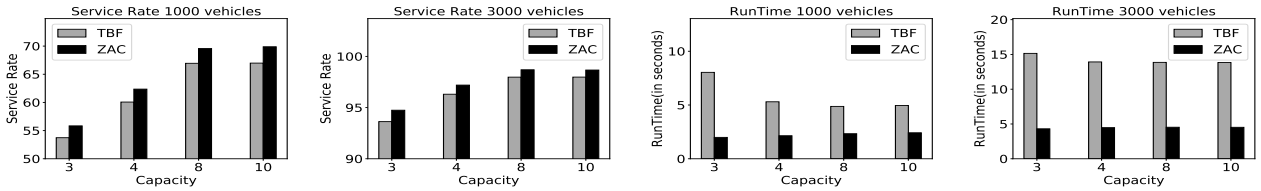


Figure 8: Comparison of ZAC and TBF on NYDataset for τ=180 seconds, λ=600 seconds and $\Delta = 60$ seconds

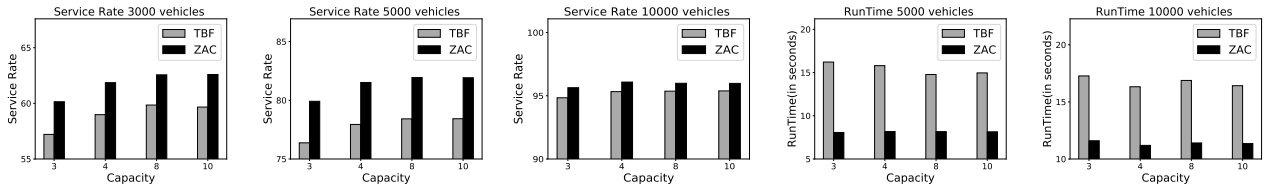


Figure 9: Comparison of of ZAC and TBF on Dataset1 for $\tau = 180$ seconds, $\lambda = 600$ seconds and Δ= 60 seconds

M=4 (with zone sizes 0,60,120,300), we get the right trade-off between computational complexity and solution quality. Therefore, we use this configuration for ZAC to compare with TBF.

5.3 Real world datasets

In this section, we compare the service rate and runtime of TBF and ZAC by varying different parameters on two real world datasets.

Effect of change in vehicle capacity (μ) and number of vehicles ($|V|$): Figure 8 and Figure 9 show the service rate and runtime comparison of TBF and ZAC for NYDataset and Dataset1 respectively at 8am (Peak time) .

Here are the key observations when vehicle capacity is changed for a fixed number of vehicles:

(1) Service rate obtained by ZAC is more than TBF for both datasets. For capacity 4 with 1000 vehicles for NYDataset, the service rate obtained by ZAC is 2.29% more than the service rate obtained by TBF and for capacity 10 we obtain a gain of 2.89%. On the other hand for Dataset1 for capacity 4 with 5000 vehicles, the service rate obtained by ZAC is upto 4% more than the service rate obtained by TBF. On Dataset1, we do not observe the increase in service rate beyond capacity 4 due to large size of the network and longer travel times which allows fewer requests to be paired. (2) While both ZAC and TBF can compute a solution in less than 20 seconds, the time taken by ZAC is much less than TBF.

For the change in the number of vehicles, we make the following observations:

(1) On Dataset1, the difference in the service rate obtained by ZAC and TBF increases as the number of vehicles increases from 3000 to 5000. This is because, TBF limits the number of vehicles considered for each request to 30, so the number of requests missed due to this limit will be more for higher number of vehicles. But on further increasing the number of vehicles to 10000, the gap between ZAC and TBF reduces. This is because, when more vehicles are available, it reduces the need of generating all combinations. On NYDataset, the difference between service rate obtained by ZAC and TBF is maximum for 1000 vehicles.

Effect of change in value of Δ: We compare the service rate and runtime of algorithms for different values of Δ (Figure 10). Here are the key observations: (1) Service rate increases as the value of Δ increases. This is because more requests are available at each decision epoch which allows grouping more requests together.(2) Difference between service rate of ZAC and TBF decreases as Δ increases. This is because assignment of vehicles to requests at each decision epoch restricts the future combinations for vehicle so it is more beneficial to explore all combinations at each decision epoch. (3) The time taken by TBF is much more than ZAC for larger Δ values due to the presence of more number of requests at each decision epoch.

Effect of time of the day: We compare the effect of time of day on the performance of algorithms (Figure 11). Here are the key observations: (1) The service rate of ZAC is more than TBF in each time interval. (2) The difference between service rate of ZAC and TBF is more during non-peak hours (3pm and 12am) as there are less requests available at each decision epoch, so as opposed to peak time where there are more possibility of grouping requests across decision epochs, at non-peak times it is advantageous to explore more combinations at a single decision epoch.

Effect of change in values of τ and λ: We show the service rate and runtime results for different values of τ and λ in Figure 12. Irrespective of the delay constraints, service rate obtained by ZAC is either more or same as TBF and the runtime of ZAC remains less than TBF in all cases.

On real datasets, ZAC obtains up to 4% gain in service rate over TBF across different parameter values. Typically, even a 0.5% gain is considered significant on real taxi datasets (as shown by a real car aggregation company (Xu et al. 2018)), so 4% represents a significant gain.

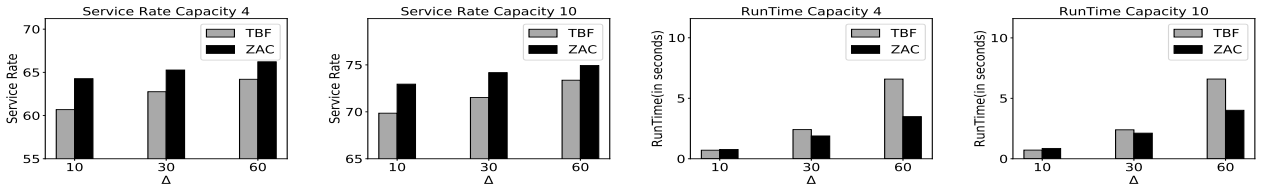


Figure 10: Comparison of TBF and ZAC for NYDataset for 1000 vehicles and varying values of Δ, τ = 300, λ = 600 seconds

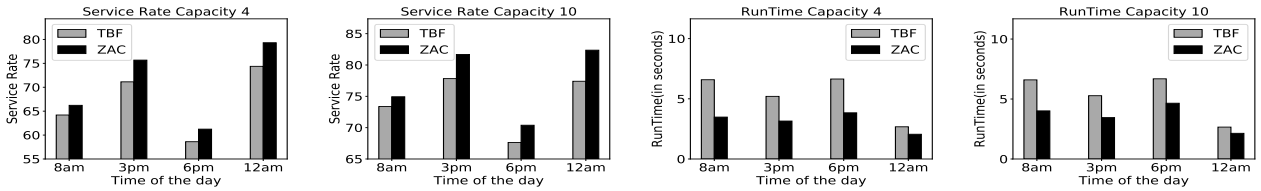


Figure 11: Comparison of TBF and ZAC for NYDataset for 1000 vehicles and different time of the day. τ = 300, λ = 600 seconds

5.4 Synthetic Dataset

The real world taxi datasets can not capture the scenarios for on demand shuttle services (Shotl 2018; Beeline 2016; Grab 2018) having a small set of pickup/drop-off points in a city. These involve scenarios where many requests can be combined at each decision epoch. We represent these scenarios by simulating the case of first and last mile transportation in the synthetic network (details provided in experimental setup), where there are multiple requests at each decision epoch with either identical pickup location and nearby drop-off locations or identical drop-off locations and nearby pickup locations resulting in higher possibility of having large number of request combinations at a decision epoch.

The gain obtained by ZAC over TBF is even more significant in these scenarios as TBF will not be exploring all relevant combinations while ZAC can explore more combinations by using zone paths. We compare the service rate obtained by TBF and ZAC with different number of vehicles and different capacities and make following observations:
(1) We observe that with 500 vehicles and capacity 10, we can obtain 20.8% gain in service rate. (2) With 1000 vehicles, ZAC can serve 91% of the requests available while TBF could serve only 75% requests. The gain reduces on increasing the number of vehicles as in presence of sufficient vehicles, we can serve the requests irrespective of current assignments.

These results clearly demonstrate that ZAC is able to consider significantly more trips than TBF.

6 Conclusion

In this paper, we presented a zone path based clustering approach which can efficiently perform ridesharing for higher capacity vehicles. The experimental comparison on real world and synthetic datasets show that our approach can outperform the current best approach (used even by taxi and car aggregation companies like Grab and Lyft) in terms of both runtime and solution quality. In future, we would be extending this work to consider future demand to further improve the solution quality.

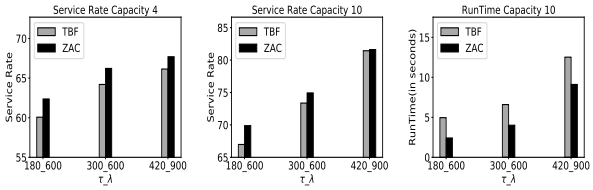


Figure 12: NYDataset - 1000 vehicles, Δ=60 seconds.

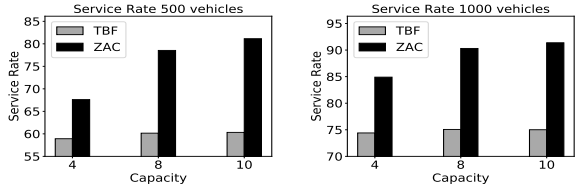


Figure 13: Synthetic Dataset with $\tau = 120$,$\lambda = 240$ and $\Delta = 60$ seconds

7 Acknowledgements

This work was partially supported by the Singapore National Research Foundation through the Singapore-MIT Alliance for Research and Technology (SMART) Centre for Future Urban Mobility (FM). We thank Sanket Shah for providing valuable comments which greatly improved the paper.

References

Agatz, N.; Erera, A. L.; Savelsbergh, M. W.; and Wang, X. 2011. Dynamic ride-sharing: A simulation study in metro atlanta. *Procedia-Social and Behavioral Sciences* 17:532–550.

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

Beeline. 2016. Beeline singapore - book seat on the buses. https://www.beeline.sg/.

Bei, X., and Zhang, S. 2018. Algorithms for trip-vehicle assignment in ride-sharing.

Bertsimas, D.; Jaillet, P.; and Martin, S. 2018. Online vehicle routing: The edge of optimization in large-scale applications. *Operations Research*.

Boeing, G. 2017. Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems* 65:126–139.

Browns, T. 2016. Match making in lyft line. https://eng.lyft.com/matchmaking-in-lyft-line-9c2635fe62c4.

Chen, L.; Gao, Y.; Liu, Z.; Xiao, X.; Jensen, C. S.; and Zhu, Y. 2018. Ptrider: a price-and-time-aware ridesharing system. *Proceedings of the VLDB Endowment* 11(12):1938–1941.

Cheng, P.; Xin, H.; and Chen, L. 2017. Utility-aware ridesharing on road networks. In *Proceedings of the 2017 ACM International Conference on Management of Data*, 1197–1210. ACM.

Dube-Rioux, L.; Schmitt, B. H.; and Leclerc, F. 1989. Consumers' reactions to waiting: when delays affect the perception of service quality. *ACR North American Advances*.

Dutta, C. 2018. When hashing met matching: Efficient search for potential matches in ride sharing. *arXiv preprint arXiv:1809.02680*.

Grab. 2018. Grab shuttle plus - on demand shuttle service. https://www.grab.com/sg/shuttleplus/.

Hasan, M. H.; Van Hentenryck, P.; Budak, C.; Chen, J.; and Chaudhry, C. 2018. Community-based trip sharing for urban commuting.

Huang, Y.; Bastani, F.; Jin, R.; and Wang, X. S. 2014. Large scale real-time ridesharing with service guarantee on road networks. *Proceedings of the VLDB Endowment* 7(14):2017–2028.

Lin, K.; Zhao, R.; Xu, Z.; and Zhou, J. 2018. Efficient large-scale fleet management via multi-agent deep reinforcement learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1774–1783. ACM.

Lowalekar, M.; Varakantham, P.; and Jaillet, P. 2018. Online spatio-temporal matching in stochastic and dynamic domains. *Artificial Intelligence* 261:71–112.

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

Maister, D. H., et al. 1984. *The psychology of waiting lines*. Harvard Business School Boston, MA.

NYYellowTaxi. 2016. New york yellow taxi dataset. http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml.

Parragh, S. N.; Doerner, K. F.; and Hartl, R. F. 2008. A survey on pickup and delivery problems. *Journal für Betriebswirtschaft* 58(1):21–51.

Pelzer, D.; Xiao, J.; Zehe, D.; Lees, M. H.; Knoll, A. C.; and Aydt, H. 2015. A partition-based match making algorithm for dynamic ridesharing. *IEEE Transactions on Intelligent Transportation Systems* 16(5):2587–2598.

Puterman, M. L. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Ritzinger, U.; Puchinger, J.; and Hartl, R. F. 2016. A survey on dynamic and stochastic vehicle routing problems. *International Journal of Production Research* 54(1):215–231.

Ropke, S., and Cordeau, J.-F. 2009. Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science* 43(3):267–286.

Ropke, S.; Cordeau, J.-F.; and Laporte, G. 2007. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks* 49(4):258–272.

Santi, P.; Resta, G.; Szell, M.; Sobolevsky, S.; Strogatz, S. H.; and Ratti, C. 2014. Quantifying the benefits of vehicle pooling with shareability networks. *Proceedings of the National Academy of Sciences* 111(37):13290–13294.

Shotl. 2018. Shotl on demand shuttles. https://shotl.com/.

Tang, M.; Ow, S.; Chen, W.; Cao, Y.; Lye, K.-w.; and Pan, Y. 2017. The data and science behind grabshare carpooling. In *Data Science and Advanced Analytics (DSAA), 2017 IEEE International Conference on*, 405–411. IEEE.

Tong, Y.; Zeng, Y.; Zhou, Z.; Chen, L.; Ye, J.; and Xu, K. 2018. A unified approach to route planning for shared mobility. *Proceedings of the VLDB Endowment* 11(11):1633–1646.

Widdows, D.; Lucas, J.; Tang, M.; and Wu, W. 2017. Grabshare: The construction of a realtime ridesharing service. In *Intelligent Transportation Engineering (ICITE), 2017 2nd IEEE International Conference on*, 138–143. IEEE.

Xu, Z.; Li, Z.; Guan, Q.; Zhang, D.; Li, Q.; Nan, J.; Liu, C.; Bian, W.; and Ye, J. 2018. Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 905–913. ACM.

Yang, J.; Jaillet, P.; and Mahmassani, H. 2004. Real-time multivehicle truckload pickup and delivery problems. *Transportation Science* 38(2):135–148.

Zheng, L.; Chen, L.; and Ye, J. 2018. Order dispatch in price-aware ridesharing. *Proceedings of the VLDB Endowment* 11(8):853–865.