A Greener Transportation Mode: Flexible Routes Discovery from GPS Trajectory Data*

Favyen Bastani
University of North Texas
favyenbastani@my.unt.edu

Yan Huang
University of North Texas
huangyan@unt.edu

Xing Xie
Microsoft Research Asia
Xing.Xie@microsoft.com

Jason W. Powell
University of North Texas
jason.powell@unt.edu

ABSTRACT

We propose a flexible mini-shuttle like transportation system called *flexi*, with routes formed by analyzing passenger trip data from a large set of taxi trajectories. The usage of public transportation is declining as often it no longer matches with individual needs. Thus, the flexi system provides a transportation mode in between buses and taxis so that inconvenience in switching to the system can be minimized overall. To generate flexi routes, we propose a two-phase approach. In the first phase, a fast diameter-constrained agglomerative clustering algorithm is developed and applied to the set of trips derived from the GPS data. This phase identifies a set of heavily traveled spatio-temporal trip clusters called hot lines. In the second phase, a directed acyclic graph is constructed from the hot lines. Then, an optimal single flexi route discovery algorithm on graph searching is proposed. Multiple routes are discovered by iteratively applying the single routing algorithm. Extensive experiments using a large set of real taxi trajectory data show that the flexi system can save a large percentage of trip mileage.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Spatial databases and GIS

1. INTRODUCTION

Public transportation offers a means to decrease traffic congestion and fuel consumption by ride sharing. However, the usage of public transportation is declining as public transportation routes often no longer match with individual needs and personal automobiles grow more appealing. Thus, transportation system users increasingly are switching from public transportation methods to private automobiles, which can be environmentally damaging and lead to greater traffic congestion. Public transportation must adapt in order to supply modern demands. According to the American Public Transportation Association (APTA) 2011 Public Transportation Factbook, from 2008 to 2009, the total number of passenger trips using public transportation dropped from 10.5 to 10.4 billion [1]. Thus, establishing more personalized public transportation will help reverse the general trend of switching from public to private transportation.

Related work includes moving object clustering, trajectory clustering, motion path and general traffic pattern discovery, and routing. In moving object clustering, live spatio-temporal data from moving objects are processed to identify events such as cluster joining and splitting. Two approaches are rectangular bounding [6] and object movement dissimilarity [5]. Trajectory clustering involves clustering data sequences dependent on some incremental variable, usually time; in [3], this is accomplished with probabilistic clustering. In [7], hot motion paths are efficiently extracted from location-aware device data by lowering communication overhead with processing on the devices. Although these data are useful for certain purposes, in transportation route formation we need a finer time granularity, similarity based on source and destination (including time), and highly specific traffic patterns (to minimize the distance passengers must walk before being picked up). Routing algorithms have been developed [8, 2], but not based on actual raw taxi trip data. Such data has been used before for other purposes; in [9], fast routes to a certain destination are developed for taxis drivers using a time-dependent landmark graph and Variance-Entropy-Based Clustering.

This paper attempts to mine combinable trips from taxi GPS trajectories and suggest a flexible mini-shuttle like transportation system that lies between bus and subway transportation, which are slower due to stops, and taxis, which cause greater traffic congestion and pollution. The main challenges of devising such a system are deciding which trips are combinable and designing effective routing algorithms. Specifically, we make the following contributions: (1) adaptation of the basic agglomerative hierarchical clustering algorithm [4] into a fast diameter-constrained clustering-based approach to combine multiple taxi trips (hot lines) with a service guarantee; (2) a method to allow the hot lines to form a directed acyclic graph and an efficient routing algorithm for the new transportation system; (3) experimental validation of the proposed algorithm on a large real taxi trajectory dataset.

*This work was partially supported by the National Science Foundation under Grant No. IIS-1017926.

ACM SIGSPATIAL GIS '11, November 1-4, 2011. Chicago, IL, USA

2. METHODOLOGY

2.1 Problem Definition

We are interested in creating a flexible mini-shuttle system consisting of vehicles called flexis that operate in a hybrid mode between taxis and buses. A flexi operates throughout the day attempting to transport multiple passengers per trip. The flexi's schedule is available ahead of time but does not contain short routes recurring with a set frequency currently used in bus systems. Because it has a smaller capacity than a bus, a flexi does not require marked stops.

Definition [*Dratio* and *Eratio*]: The distance ratio, *Dratio*, is defined as $Dratio = \frac{D_{flexi} + D_{\widetilde{flexi}}}{D_{taxi}}$, where D_{flexi} is the total distance that flexis travel based on the flexi schedule, $D_{\widetilde{flexi}}$ is the total distance of trips not satisfied by a flexi in a system with flexis and taxis, and D_{taxi} is the total distance of all trips in the taxi system. The combination or efficiency ratio *Eratio* is defined as $Eratio = \frac{D_{flexi}}{D_{taxi} - D_{\widetilde{flexi}}}$.

Eratio is the combination factor of trips (the average number of flexi trips that one original taxi trip has been combined into). *Dratio* and *Eratio* are defined such that both are better when lower and both should be between 0 and 1 for reasonable generated trips.

Given: A set $S = \{s_1, s_2, ..., s_n\}$ of taxi trips of size n. Each trip s_i contains a set of six attributes that define the trip, $\{s_i.sx, s_i.sy, s_i.st, s_i.ex, s_i.ey, s_i.et\}$. $s_i.sx$ and $s_i.sy$ are the starting coordinates of s_i, and $s_i.st$ is the trip's starting time. $s_i.ex$ and $s_i.ey$ are the ending coordinates, and $s_i.et$ is the ending time.

Goal: To combine the trips in S and establish routes for the flexi system. The flexi schedule should maximize the demand satisfied by flexis.

Objective: To minimize *Dratio* while maintaining an acceptably low *Eratio*.

The meaning of an acceptable *Eratio* is that the following must be true: $Eratio < \frac{C_{taxi}}{C_{flexi}}$, where C_{taxi} is the operating cost for one taxi and C_{flexi} is the operating cost for one flexi under the same conditions.

2.2 Hot Line Discovery

We would like to find clusters of trips that can be combined. The trips to be combined should share similar starting times, durations, origins, and destinations. Here, we develop a general hierarchical clustering algorithm; thus, we use $s_i.ai$ as ith attribute of s_i.

Our goal is to ensure that trips that form a hot line have certain space and time service guarantees. We set the limit on the distance a node (trip) may be from its cluster center to θ, and define distance as the normalized Euclidean distance between its attributes. Normalization is manually performed because distance and time may vary in the data. The algorithm supports normalization of the l attributes using a set of weights (modifiable to account for differences between attributes) , $W = \{w_1, w_2, ..., w_l\}$. Then, the distance between two nodes s_a and s_b with l attributes is $d(s_a, s_b) = \sqrt{((s_a.a1 - s_b.a1)/w_1)^2 + ... + ((s_a.al - s_b.al)/w_l)^2}$

Many hierarchical clustering algorithms are available. Because we are only interested in clusters with diameters less than the threshold θ, we develop a distance-constrained, agglomerative, complete-linkage hierarchical clustering algorithm. The algorithm only keeps track of the cluster pairs with distance less than θ.

When clustering begins, each trip s_i is in a separate cluster; thus, there are n clusters initially and the size of each cluster c_i, $|c_i| = 1$. During each iteration, the two closest clusters c_a and c_b are merged (i.e., each element of c_b is removed from c_b and inserted into c_a). The distance between clusters is defined as $d(c_a, c_b) = \max\{d(i, j) | i \in c_a, j \in c_b\}$

Initially, the distance $d(c_i, c_j)$ is calculated between every two clusters c_i and c_j, and distances less than θ are stored along with the nodes into a linked list D. The element of D at x contains the indexes of the nodes it represents, $D_x.a$ and $D_x.b$, and the distance between those nodes, $D_x.d$. Thus, $d(c_{D_x.a}, c_{D_x.b}) = D_x.d$. We also use the notation $D_x = \{D_x.a, D_x.b, D_x.d\}$. D is sorted initially in ascending order and is maintained in sorted order after clusters are merged.

For each cluster c_i, all D_x containing c_i in the node pair are stored into a neighbor list $c_i.m$. $c_i.m$ contains references to the linked list, and is used later to quickly update D after a merge of clusters happens.

D is then read sequentially. For each D_x, clusters $D_x.a$ and $D_x.b$ are first merged into $D_x.a$, i.e. $D_x.a$ absorbs $D_x.b$. D must then be updated as the cluster $D_x.b$ no longer exists. For a cluster c, if the pair $(D_x.a, c)$ is in $D_x.a.m$ but the pair $(D_x.b, c)$ is not in $D_x.b.m$, then $(D_x.a, c)$ can be safely removed from D. This is because if the distance from c to $D_x.b$ is greater than θ (not exist in D), then the maximum distance from the merged cluster to c must be greater than θ. Similarly, if the pair $(D_x.b, c)$ is in $D_x.b.m$ but the pair $(D_x.a, c)$ is not in $D_x.a.m$, then $(D_x.b, c)$ can be safely removed from D. Otherwise, D contains both $(D_x.a, c)$ at index y and $(D_x.b, c)$ at index z. If $D_y.d < D_z.d$, then we set $D_z = \{D_y.a, D_y.b, D_z.d\}$ and D_y is deleted. Otherwise, D_z is deleted without update to D_y.

Hierarchical clustering continues until D contains only the root of the linked list that does not represent an actual distance pair. The ending number of clusters is equal to $n - e$, where e is the number of iterations elapsed. After clustering, the final clusters are arranged such that the lowest index is 1 and the highest index is $n - e$. Then, we can calculate the average attributes for each trip cluster. For a cluster of taxi trips c_i, let $c_i.ak$ be the average value of the nodes' $s_i.ak$.

In the taxi trip problem, s_i contains the six attributes that define the trip. $s_i.a1$ and $s_i.a2$ are $s_i.sx$ and $s_i.sy$, $s_i.a3$ is $s_i.st$, $s_i.a4$ and $s_i.a5$ are $s_i.ex$ and $s_i.ey$, and $s_i.a6$ is $s_i.et$.

For combined spatiotemporal comparisons, we must normalize spatial and temporal distances. This is accomplished by setting $w_1 = w_2 = w_4 = w_5$ and $w_3 = w_6$. We choose one unit for the maximum radius θ (i.e., the maximum distance that a trip may be from its cluster) and base choices of weights on this radius.

2.3 Routing Algorithm

In order to identify a route that connects multiple taxi trip clusters, our routing algorithm must identify which clusters will be connected (i.e., which clusters may follow a specific cluster in a route). To do this, we must calculate the time necessary to travel from the ending point of one trip cluster to the beginning point of one cluster. Then, we can estimate the time a flexi will have to wait at a stop by subtracting this traveling time from the time in between the trip clusters.

Here, we consider taxi trips instead of data points, so we will use a similar notation as the system from subsection 2.1.

Let $t(c_i, c_j) = \frac{m(c_i,c_j)}{s}$ be the estimated traveling time from c_i to c_j, where $m(c_i, c_j)$ is the Manhattan distance from $(c_i.ex, c_i.ey)$ to $(c_j.sx, c_j.sy)$ and s is the average speed of the trip. s is estimated as 11 km/h based on averages in taxi trip data.

A cluster c_i is considered a parent of c_j if $0 < c_j.st - c_i.et - t(c_i, c_j) < M$, where M is the maximum time in minutes a flexi will wait at a stop. The set of n_i parents of cluster c_i $P(c_i) = \{p_{i,1}, p_{i,2}, ..., p_{i,n_i}\}$. The fitness of a single trip cluster is $f(c_i) = |c_i|\sqrt{(c_i.ex - c_i.sx)^2 + (c_i.ey - c_i.sy)^2}$. To find the longest route in G, we first sort the set of clusters C by starting time st. Then, we iterate through C in increasing order of st and calculate the route fitness $F(c_i) = max\{F(p_{i,1}), F(p_{i,2}), ..., F(p_{i,n_i})\} + f(c_i)$. $max\{\}$ (in the case that a taxi cluster has no parents) is defined as zero.

Once the route fitness of each cluster has been calculated, we search the set for the largest value and select the the route defined by that cluster. Additionally, a capacity-based routing algorithm is tested that limits the number of people taken on any trip cluster in the route to L, lowering operating costs. We also implement a fast greedy algorithm that selects the next point in a route based on maximizing the weight gain, and compare the three algorithms.

3. EXPERIMENTAL SETUP

Data is collected from over seventeen thousand taxis from three companies in Shanghai for one day (May 29, 2009) consisting of entries that contain the time and GPS coordinates. The data consists of 432,327 trips with each trip representing the transfer of passengers from a starting location to a destination. This spatio-temporal data can be used to improve traffic congestion in dense urban areas by optimizing flexi routes, which may replace existing bus routes.

We evaluate the routing algorithms in solution quality and scalability. Solution quality is measured by *Dratio* and *Eratio*: *Dratio* should be minimized while maintaining an acceptable *Eratio*. The threshold for an acceptable *Eratio* is unknown because the operating cost ratio is unknown and depends on the specific vehicles chosen. Scalability is measured by execution time with respect to the number of cluster trips the routing algorithm must consider.

For hierarchical clustering, we set a reasonable maximum walking distance of 500 m and a maximum waiting time of 12 min. As these are both maximum amounts, they should be equivalent after normalization, thus can be handled by $\theta = 1$, $w_1 = w_2 = w_4 = w_5 = 500$, and $w_3 = w_6 = 12$. We additionally test maxima of 250 m and 6 min. In the routing algorithm, the maximum time a flexi will wait at a stop M is set to 15 min.

4. RESULTS

First, we conduct experiments to determine a reasonable capacity to use for the capacity-based routing algorithm. This is accomplished by analyzing results from the normal routing algorithm and estimating the loss in solution quality by limiting the number of passengers in each step of the flexi route found (the optimal route may be different for a capacity-based routing algorithm because the decrease in F is lower for flexi routes with higher passenger counts).

Fig. 1 shows the distance and passenger fractions (y axis)

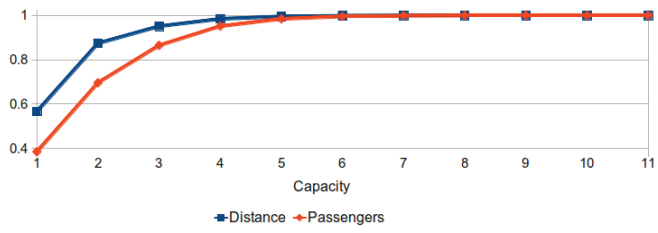

Figure 1: Distance and passenger fractions at different capacities

at different capacities (x axis) with 6,000 flexis. The distance fraction is $\frac{\text{limited route fitness}}{\text{original route fitness}}$, and the passenger fraction is $\frac{\text{limited number of passengers}}{\text{original number of passengers}}$, where the number of passengers is the sum of $|c_i|$ in each cluster of the route. Note that the clusters along routes had no more than eleven passengers.

Based on the graph, we choose six as the capacity for further experimentation. In graphs, routing indicates the original routing algorithm, capacity-based routing indicates the limited capacity routing algorithm, and greedy indicates the simple greedy algorithm.

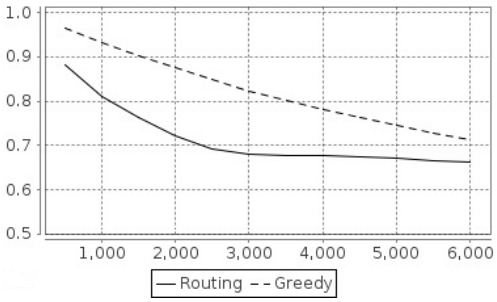

Figure 2: *Dratio* with 250 m maximum distance and 6 min maximum waiting time

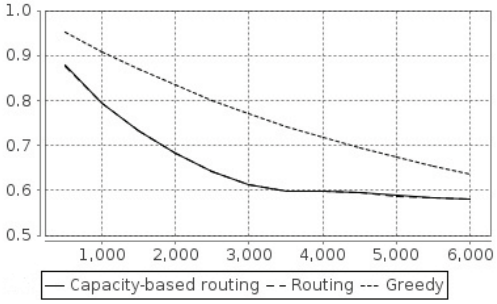

Figure 3: *Dratio* with 500 m maximum distance and 12 min maximum waiting time

Figures 2 and 3 show that with looser weights, the improvement in *Dratio* increases. Also, in both graphs, the slope of the curve gradually decreases becauses the routing algorithms select the best routes first. In Figure 3, the capacity-based routing and original routing lines cannot be distinguished, suggesting that capacity-based routing is favorable with almost identical *Dratio* but significantly lower total operating costs: the required capacity is reduced from

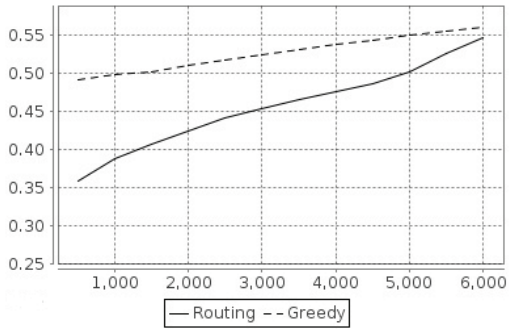

Figure 4: *Eratio* with 250 m maximum distance and 6 min maximum waiting time

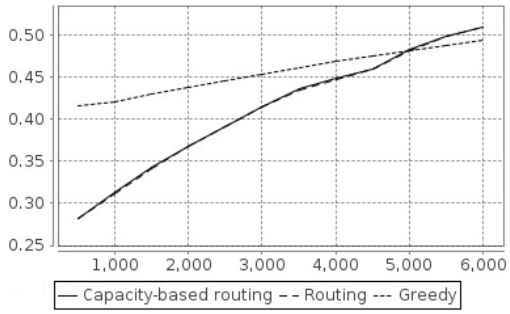

Figure 5: *Eratio* with 500 m maximum distance and 12 min maximum waiting time

eleven passengers to six passengers (not counting a driver). Lower operating costs imply lower fuel consumption.

The exact *Dratio* at 6,000 flexis in Fig. 3 for the optimal routing algorithm is 0.579, and for the greedy routing algorithm is 0.637. This suggests that with flexis in place vehicles will have to travel approximately 57.9% of the distance that taxis originally traveled. The capacity-based routing algorithm has a *Dratio* of 0.580, showing that there is indeed some loss in solution quality.

Eratio, like *Dratio*, generally improves with less restrictive maximum combination values. In Figure 5, the greedy algorithm actually achieves a lower *Eratio* than the optimal routing algorithm. The rate of deterioration in *Eratio* for the greedy algorithm is slower because the first flexi route solutions it identifies are not optimal, leaving more good solutions to be found later.

Table 1 shows the execution time of the routing algorithm and greedy algorithm with different numbers of trip clusters. The algorithms are tested with a maximum distance of 500 m and a maximum waiting time of 12 minutes, and are used to generate routes for 6,000 flexis. The number of trip clusters represent 50%, 75%, and 100% of the entire dataset

# Trip Clusters	Routing time (min)	Greedy time (min)
87167	142.6	25.56
130750	485.5	59.08
174333	1022	122.8

Table 1: Routing algorithm time with increasing number of trip clusters

for those thresholds. 25%, or 43,583 trip clusters, cannot be tested because there is too little total data ($D_{\widetilde{flexi}} = 0$ before 6,000 flexi routes are generated).

5. CONCLUSIONS

In this paper, we establish a two-phase approach for forming public transportation routes from GPS trip data. We develop a highly efficient hierarchical clustering algorithm that considers only distances below a threshold to identify heavily traveled trips and a scalable routing algorithm to form a route from the directed graph of trips. Our method considers spatio-temporal data, allowing it to handle data during a timespan in which traffic flow may vary greatly, and can be executed with different parameters to derive public transportation routes in different situations. Experiments show that our routing algorithm outputs significantly better results than a simple, fast greedy algorithm, and that the flexi system has the potential to reduce fuel consumption.

We hope to continue improving the routing algorithm, extend this approach to GPS trip data for multiple days and use recurring patterns to form routes free from day-to-day fluctuations, and envision applying this approach dynamic scenarios where flexi requests containing information including submission time, preferred coordinates, and destinations are analyzed in real time. In this scenario, after passengers request a route, they would be guaranteed a flexi by a certain time or rejected, making predetermined routes unnecessary.

6. REFERENCES

[1] Public transportation factbook. Technical report, American Public Transportation Association, 2011.

[2] Wei Fan and Randy B. Machemehl. Optimal transit route network design problem: Algorithms, implementations, and numerical results. Technical report, University of Texas at Austin Center for Transportation Research, May 2004.

[3] S. Gaffney and P. Smyth. Trajectory clustering with mixtures of regression models. In *Proceedings of the fifth ACM SIGKDD*, pages 63–72. ACM, 1999.

[4] A.K. Jain, M.N. Murty, and P.J. Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

[5] C.S. Jensen, D. Lin, and B.C. Ooi. Continuous clustering of moving objects. *IEEE Transactions on Knowledge and Data Engineering*, pages 1161–1174, 2007.

[6] Y. Li, J. Han, and J. Yang. Clustering moving objects. In *Proceedings of the tenth ACM SIGKDD*, pages 617–622. ACM, 2004.

[7] Dimitris Sacharidis, Kostas Patroumpas, Manolis Terrovitis, Verena Kantere, Michalis Potamias, Kyriakos Mouratidis, and Timos Sellis. On-line discovery of hot motion paths. *EDBT*, March 2008.

[8] P. Shrivastava and M. O'Mahony. Design of feeder route network using combined genetic algorithm and specialized repair heuristic. *Journal of Public Transportation*, 10(2):109–133, 2007.

[9] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang. T-drive: driving directions based on taxi trajectories. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 99–108. ACM, 2010.