Taxi Queue, Passenger Queue or No Queue?

A Queue Detection and Analysis System using Taxi State Transition

Yu Lu, Shili Xiang, Wei Wu
Institute for Infocomm Research, A*STAR, Singapore
{luyu, sxiang, wwu}@i2r.a-star.edu.sg

ABSTRACT

Taxi waiting queues or passenger waiting queues usually reflect the imbalance between taxi supply and demand, which consequently decrease a city's traffic system productivity and commuters' satisfaction. In this paper, we present a queue detection and analysis system to conduct analytics on both taxi and passenger queues. The system utilizes the event-driven taxi traces and the taxi state transition knowledge to detect queue locations at a coordinate level and subsequently identify 4 different types of queue context (e.g., only passengers queuing or only taxis queuing). More specifically, it adopts the novel and easy-to-implement algorithms to selectively extract taxi pickup events and their critical features. The extracted taxi pickup locations are then used to detect queue locations, and the extracted critical features are used to infer queue context. The extensive empirical evaluations, which run on daily 12.4 million taxi trace records from nearly 15000 taxis in Singapore, demonstrate the high accuracy and stability of the queue analytics results. Finally, we discuss the real world deployment issues and the gained insights from the queue analysis results.

1. INTRODUCTION

In the densely populated Asian cities (e.g., Singapore, Beijing and Taipei), relatively cheap taxi fares and large number of taxis greatly facilitate the pervasive usage of taxis by urban citizens for various purposes, such as traveling between office and home, purchasing groceries at supermarkets and visiting friends. It is relatively different from the taxi usage at many cities in US or Europe, where taxis more frequently serve airport routes and do not cover all urban districts. The taxi usage characteristics in the Asian cities easily cause that the temporal and spatial imbalance of taxi supply and demand occurs frequently: taxis would queue up for passengers due to temporarily low taxi demand but high supply nearby; passengers would queue up for taxis due to temporarily high taxi demand but low supply; in many time periods, taxis and passengers would concurrently queue up as both taxi demand and supply are high. Such queuing events usually not only reduce the productivity of an urban traffic system, but also greatly decrease the satisfaction of public commuters as well as taxi drivers. Fig. 1 illustrates a taxi queue and a passenger queue that both frequently occur in Singapore.

(a) Taxi Queue

(b) Passenger Queue

Figure 1: Different Types of Queue in Singapore

Properly and accurately detection of queue locations and queue context would benefit many parties and stakeholders. The real time queuing events information and their long-term patterns can be used in the recommendation systems for taxi drivers and commuters (e.g., suggest commuters to the nearby taxi queue locations). The information can also be used in the taxi operators' booking and dispatching systems (e.g., guide available taxis to passenger queue locations). Moreover, the government agencies need such information to understand the imbalance between taxi supply and demand, and accordingly take necessary actions (e.g., increase operating taxis or adjust taxi fares).

Motivated by the availability of abundant information in taxi traces, e.g., GPS locations and taxi states, using taxi traces to design and build a city scale queue detection and analysis system is a promising solution. However, it is an open and non-trivial problem. Firstly, taxi queuing for passengers is not simply a passenger pickup, dropoff or vehicle parking event, only the GPS coordinates and the binary taxi states (occupied or non-occupied) are not enough to capture it. Secondly, passenger queuing for taxis is even more difficult to detect, as no any direct information from the passenger side and no apparent clue in taxi traces. Thirdly, both taxi queuing and passenger queuing are highly dynamic in terms of time and locations, which not only repeatedly occur at fixed taxi stands or during peak hours.

In this paper, we present a practical system that captures

10.5441/002/edbt.2015.60

taxis and passengers queuing activities at a fine-grained scale, i.e., at the individual coordinate level rather than a region or zone, and subsequently analyzes different types of queue context. We summarize the key contributions of this work as follows:

- We propose a novel approach, using multiple taxi states and their transition information, to conduct the city scale queue analytics for both taxis and passengers.
- We design and implement a two-tier queue analytics engine, where the lower tier module detects queuing locations and the upper tier module identifies queue context based on the selected taxi pickup events and features.
- We conduct the extensive empirical evaluation of our queue analytics results before we deploy the system in the real world. We demonstrate its stability using large scale taxi traces, and its accuracy using various other data sources (e.g., landmark information, failed taxi booking data).

The rest of the paper is organized as follows: section 2 introduces the background of the dataset, and then section 3 depicts our overall system architecture and define the queue types. In sections 4 and 5, we describe our queue spot detection and queue context disambiguation modules in detail respectively. Extensive empirical evaluations are conducted in section 6, which is followed by a discussion on deployment and other issues in section 6. The related work is presented in section 8. At last, we conclude with future work in section 9.

2. TAXI STATE AND EVENT-DRIVEN LOG

2.1 Mobile Data Terminal

As part of the taxi operators' efforts on improving their quality of service, each taxi in Singapore is equipped with a specifically designed device, called mobile data terminal (MDT), which is mainly used to handle taxi bookings and monitor a taxi's real time status. More specifically, it receives taxi booking tasks from the backend service (taxi call center), and sends back taxi driver's decision (accept or reject the task) via general packet radio service (GPRS). Moreover, MDT keeps logging and updating a taxi's real time state by collecting the information from taxi meter, roof-top signs and its frontend touch screen. Fig. 2 simply depicts an MDT system on a Singapore taxi, where MDT is hardwired directly to different on-vehicle devices and provides taxi drivers a multifunctional touch screen.

2.2 Taxi State

Based on the collected real time information, the MDT device is able to precisely identify 11 different taxi states. Table 1 lists all the taxi states with their descriptions. The taxi state transitions mainly depend on the type of a taxi job. In principle, all taxi jobs can be classified into two categories: *street* job and *booking* job.

A *street* job means a taxi picks up new passengers by street hail, and the following is the typical taxi state transitions on a street job:

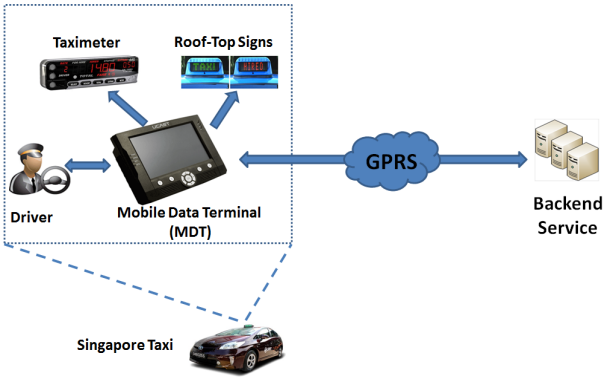

Figure 2: A simplified telematics system on a Singapore Taxi

a) a passenger hails down a taxi with *FREE* state along a road or a taxi stand.
b) the taxi driver starts the taximeter for a new trip, and meanwhile the MDT updates the taxi state to *POB*.
c) during the trip, the taxi state keeps *POB* while the MDT periodically updates the taxi GPS location.
d) the taxi is approaching the destination and the driver presses the STC button on the MDT touch screen to update the taxi state to *STC*.
e) upon arrival of the destination, the driver presses the button on the taximeter for printing the receipt, and meanwhile the MDT updates the taxi state to *PAYMENT*.
f) once the driver resets the taximeter after the passenger alights, the MDT automatically updates the taxi state to *FREE* again.

A *booking* job means a taxi picks up new passengers, who have made a booking via telephone, short message service (SMS) or mobile phone applications (apps). The typical taxi state transitions on a booking job can be described as below:

a) a passenger makes a taxi booking, and the backend service dispatches the booking information to the nearby taxis with *FREE* or *STC* state.
b) a taxi driver successfully bids the booking job by pressing the button on the MDT touch screen, and meanwhile the MDT updates the taxi state to *ONCALL*.
c) upon arrival of the booking pickup location, the MDT updates the taxi state to *ARRIVED*.
d) if the passengers do not show up within a specific time period (e.g., 15 minutes), the MDT updates the taxi state to NOSHOW first and then to *FREE* within 10 seconds.
e) if the passenger gets on the taxi in time, the MDT updates the taxi state to *POB* once the driver starts the taximeter.
f) the subsequent taxi state transitions are the same as street job's procedure, i.e., from street job's step c) to step f).

Fig. 3 illustrates a complete taxi state transition diagram, which includes the procedures of both street jobs and booking jobs.

Table 1: Taxi State and Description

Taxi State	Description
FREE	Taxi unoccupied and ready for taking new passengers or bookings
POB	Passenger on board and taximeter running
STC	Taxi soon to clear the current job and ready for new bookings
PAYMENT	Passenger making payment and taximeter paused
ONCALL	Taxi unoccupied, but accepted a new booking job
ARRIVED	Taxi arrived at the booking pickup location and waiting for the passenger
NOSHOW	No passenger showing up and the booking canceled soon
BUSY	Taxi driver temporarily unavailable due to a personal reason
BREAK	Taxi on a break and driver logged on MDT
OFFLINE	Taxi on a break and driver logged off from MDT
POWEROFF	MDT shut down and not working

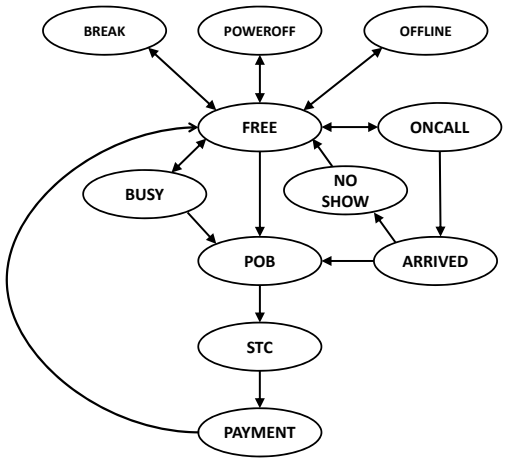

Figure 3: Taxi State Transition Diagram

2.3 MDT Log

As the central processing device on a taxi, MDT keeps updating and tracking any changes of taxi state and other critical information, e.g., GPS location, vehicle speed and taxi fares. The MDT logging module writes all such information to its local storage, and meanwhile selectively and periodically sends them to the backend service via GPRS. The MDT logging frequency is not fixed by default, and a logging action is triggered by the taxi state changes, GPS location updates and a few other critical vehicle events. Different from the traditional GPS localizer traces, the MDT log module adopts the event-driven logging mechanism, which are explicitly driven by the 11 taxi state transition events. Therefore, the MDT log captures much more accurate and abundant information than the traditional GPS traces, and accordingly provides more opportunities to discover and understand activities of both taxis and passengers.

We use the MDT log from a large local taxi operator, and select its 6 fields: timestamp, taxi ID, GPS location, instantaneous taxi speed and taxi state. Table 2 gives the selected fields in the MDT log and a sample record.

Table 2: Selected Fields of MDT Log with a Sample

Timestamp	Taxi ID	Longitude	Latitude	Speed	Taxi State
01/08/2008 19:04:51	SH0001A	103.7999	1.33795	54	POB

3. SYSTEM OVERVIEW

The system block diagram with the proposed Queue Analytic Engine is illustrated in Fig 4, and it mainly consists of two core modules:

- Queue Spot Detection Module: this module is the component for detecting queue locations (spots) based on the selected taxi pickup events. An algorithm is specifically designed for this module to extract pickup event sub-trajectories, which uses both the taxi state transition knowledge and the taxi instantaneous speed. In order to detect stable queue spots, it requires a relatively long-term historical dataset, e.g., a full day's MDT logs, from a large number of taxis.

- Queue Context Disambiguation Module: this module is the component for identifying different queuing situations, as defined in Table 3, at a queue spot. Based on the two newly proposed algorithms, the module firstly fetches the required features from the input of pickup event sub-trajectories, and then uses the features to resolve distinct queue types. The taxi state transition knowledge are used to accurately capture different time points in pickup events. This module mainly runs on a relatively short-term historical dataset for analyzing queue context transition patterns and queue types.

As mentioned earlier, Table 3 defines the four queue types, i.e., C_1 to C_4. The queue type C_1 is both taxi queue and passenger queue concurrently occur at the given queue spot, which indicates taxi demand and supply are presently both high. The queue type C_2 is only passenger queue, C_3 is only taxi queue, and C_4 is neither taxi queue nor passenger queue at the given queue spot.

In this paper, "queue" refers to a stable number of waiting entities during a specific time period, which indicates that the average arrival rate exceeds the average service rate. To

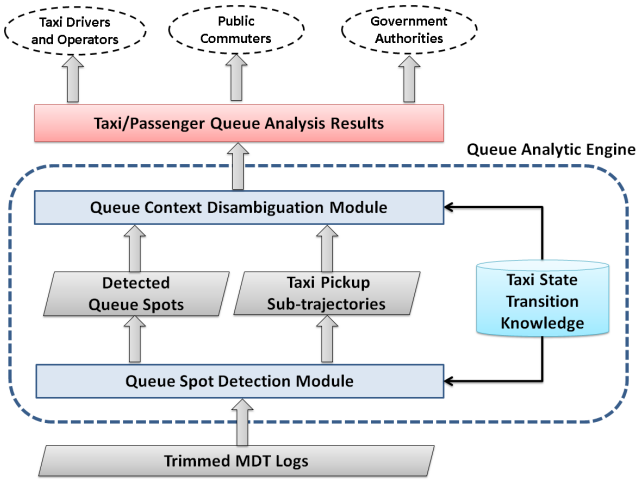

Figure 4: System Block Diagram of Queue Analytic Engine

Table 3: Four Types of Queue Context

Queue Type	Passenger Queue	No Passenger Queue
Taxi Queue	C_1	C_3
No Taxi Queue	C_2	C_4

clarify the above described queue types, we define taxi queue and passenger queue as below:

- Taxi Queue: One available taxi or more steadily awaiting for taking new passengers at a queue spot during a given time period.
- Passenger Queue: One passenger or more steadily awaiting for taxis at a queue spot during a given time period.

Note that no taxi queue or no passenger queue only means no stable waiting taxis or passengers, and it does not necessarily mean no any taxis waiting for a short time or passengers quickly getting taxis. Moreover, for both the taxi queue and the passenger queue, we do not impose any assumptions on the queue shapes and service modes, but only the first-come first-served (FIFO) discipline.

4. QUEUE SPOT DETECTION

It seems that queue spots can be easily detected by clustering the most frequent taxi pickup/dropoff locations and taxi parking locations. However, such a straightforward approach has difficulties due to two reasons. Firstly, a high proportion of *quick* pickup and dropoff events occur at any non-restricted locations in the city, and it would easily result the entire road rather than a small queuing area being a cluster. Secondly, a frequent taxi parking location is not necessarily a taxi queue spot for passengers, and thus the specific taxi state transitions need to be considered and checked.

We therefore only consider such pickup events: a taxi with an unoccupied state parks for a time of period and then departs with an occupied state. We propose a new algorithm to extract such slow pickup events, and we then determine taxi queue spots by clustering the most frequent locations of the extracted pickup events.

4.1 Preliminary

We firstly define and clarify several important terms which are used in the following sections.

Definition 1. Individual taxi's trajectory $\Re$: A temporally ordered sequence of the trimmed MDT log records from one taxi, i.e., $p_1 \rightarrow p_2 \rightarrow \cdots \rightarrow p_n$, where p_i $(1 \leq i \leq n)$ is the tuple containing the taxi state $p_{i.state}$, instantaneous speed $p_{i.speed}$, latitude coordinate $p_{i.lat}$, longitude coordinate $p_{i.lon}$ and timestamp $p_{i.ts}$.

Definition 2. Individual taxi's sub-trajectory $R(s, e)$: A segment of an individual taxi's trajectory, i.e., $p_s \rightarrow p_{s+1} \rightarrow \cdots \rightarrow p_e$, where $1 \leq s < e \leq n$.

Definition 3. Individual taxi's sub-trajectory set ω: A collection of an individual taxi's sub-trajectories, i.e., $\{R^k | k = 1, 2, \cdots\}$, where $R^k = R(s_k, e_k)$.

Definition 4. Multiple taxis' sub-trajectory set W: A collection of multiple taxi's sub-trajectory sets, i.e., $\{\omega^j | j = 1, 2, \cdots\}$, where ω^j is the j^{th} taxi's individual sub-trajectory set.

Based on the taxi state descriptions in Table 1, we classify the taxi states into three state sets:

Definition 5.1 Taxi occupied state set Θ: { POB, STC, PAYMENT }.

Definition 5.2 Taxi unoccupied state set Ψ: { FREE, ONCALL, ARRIVED, NOSHOW }.

Definition 5.3 Taxi non-operational state set Λ: { BREAK, OFFLINE, POWEROFF }.

The *BUSY* state is a special state, and we do not assign it into the above defined three taxi state sets. We will discuss it separately in the subsequent sections.

4.2 Pickup Event Extraction

In order to detect each individual taxi's slow pickup events, we propose a simple and practical algorithm, called pickup extraction algorithm (PEA): its input is an individual taxi's trajectory $\Re$ and output is the sub-trajectory set ω of the required taxi pickup events. The basic idea behind the PEA algorithm is that a slow taxi pickup event normally has at least two consecutive low speed records (e.g., below 10 km per hour) during the period of moving forward in the waiting line. Meanwhile, a pickup event shows certain taxi state transitions in the corresponding sub-trajectories, e.g., from *FREE* to *POB*. The complete algorithm is shown in Algorithm 1.

Algorithm 1 Pickup Extraction Algorithm

Input: A taxi's trajectory $\Re$ and speed threshold η_{sp}.
Output: The sub-trajectory set ω.

```
1: δ1 ← false; δ2 ← false; k ← 1;
2: for i = 1 → n do
3:   if p_{i.state} ∉ Λ then
4:     if p_{i.speed} ≤ η_sp and δ1=false and δ2=false then
5:       δ1 ← true;
6:     else if p_{i.speed} ≤ η_sp and δ1=true and δ2=false
       then
7:       R^k.Add(p_{i−1}); R^k.Add(p_i); δ2 = true;
8:     else if p_{i.speed} ≤ η_sp and δ1=true and δ2=true
       then
9:       R^k.Add(p_i);
10:    else if p_{i.speed}>η_sp and δ1=true and δ2=false then
11:      δ1 ← false;
12:    else if p_{i.speed}>η_sp and δ1=true and δ2=true then
13:      if p_{s_k.state} ∈ Θ and p_{e_k.state} ∈ Ψ then
14:        goto TAG1;
15:      else if p_{s_k.state}=FREE and p_{e_k.state}=ONCALL
         then
16:        goto TAG1;
17:      else if the taxi states in R^k never change then
18:        goto TAG1;
19:      else
20:        ω.Add(R^k); k ← k + 1;
21:      end if
22:    else
23:      goto TAG1;
24:    end if
25:  else
26:    TAG1: R^k ← ∅; δ1 ← false; δ2 ← false;
27:  end if
28: end for
```

The proposed PEA algorithm firstly filters out the sub-trajectories with any of the non-operational taxi state, i.e., $p_{i.state} \notin \Lambda$. After that, it sets the low speed flag δ_1 to *true* when the first low speed is detected, i.e., the taxi speed falls below (or equal to) the speed threshold η_{sp}. Only when the subsequent taxi speed also falls below the speed threshold, it starts to add the tuple p_i into an empty sub-trajectory R^k and repeats such an adding action until the speed becomes bigger than the threshold again. Thus, all the extracted sub-trajectories have at least two tuples with the speeds below the given threshold.

Finally, PEA adds the new R^k into the sub-trajectory set ω, given R^k satisfies the following state transition constraints: 1) R^k does not start with an occupied state and end with an unoccupied state, i.e., $p_{s_k.state} \notin \Theta$ and $p_{e_k.state} \notin \Psi$, as it is simply a "passenger alight" event; 2) R^k does not start with FREE and end with ONCALL, as it means the taxi leaves for a new booking job at another location; 3) R^k has at least one-time state transition, as we need to filter out R^k caused by traffic jams or red traffic lights.

4.3 Pickup Location Clustering

Given an individual taxi's trajectory $\Re$ having multiple slow pickup events, the output of the PEA algorithm, i.e., the sub-trajectory set ω, contains a number of sub-trajectories, i.e., $\{R^k | k = 1, 2, \cdots\}$. For each sub-trajectory R^k, i.e., $p_{s_k} \rightarrow \cdots \rightarrow p_{e_k}$, we compute a central GPS location $(\bar{c}_{lat}, \bar{c}_{lon})$ by averaging their latitude coordinates and the longitude coordinates. Accordingly, we have a GPS location set $c = \{(\bar{c}^k_{lat}, \bar{c}^k_{lon}) | k = 1, 2, \cdots\}$ derived from the sub-trajectory set ω.

After running the PEA algorithm on all taxis' trajectories respectively, we have a sub-trajectory set $W = \{\omega^j | j = 1, 2, \cdots\}$, where ω^j is the j^{th} taxi's sub-trajectory set ω. Accordingly, we have the GPS location set $C = \{c^j | j = 1, 2, \cdots\}$, where c^j is the j^{th} taxi's GPS location set c.

Given all taxis' GPS location set C, we run it with the density-based clustering method DBSCAN [5], which is an effective way to discover high density clusters and remove noises. We then compute the centroid of all the found clusters, and each centroid is the detected taxi queue spot. Given the fact that people always queue for taxis at the places where taxis usually take passengers, it is reasonable to assume the detected taxi queue spots are also the possible spots where taxi passengers queue up. We thus call an obtained centroid *queue spot* rather than taxi or passenger queue spot.

The GPS location set C is normally a large dataset. When running the DBSCAN algorithm on it, we need to carefully select its parameters and strive to reduce the runtime complexity (e.g., using the R-Tree based or grid based spatial index). We will address all of these implementation issues in section 6. On the other hand, many other advanced density-based clustering methods can also be considered and introduced [13].

5. QUEUE CONTEXT DISAMBIGUATION

5.1 Wait Time Extraction

Given $W(r)$ is the extracted pickup event sub-trajectory set W for queue spot r, it consists of a large number of sub-trajectories from different taxis. For each sub-trajectory in $W(r)$, the corresponding wait time is the time interval between the wait start time and the wait end time. We therefore present a simple algorithm, called wait time extraction (WTE) algorithm: its input is $W(r)$ for a queue spot r and output is the taxi wait time set $Y(r) = \{t^m_{end} - t^m_{start} | m = 1, 2, \cdots\}$, where t^m_{end} and t^m_{start} are the wait start time and wait end time of the m^{th} taxi. The complete algorithm is shown in Algorithm 2.

For each sub-trajectory $R(s, e)$ in $W(r)$, the WTE set the wait start time t_{start} to the timestamp when the first FREE, ONCALL or ARRIVED state appears. However, if any PAYMENT is detected thereafter, it resets t_{start} to the timestamp when the subsequent FREE state appears. After the wait start time t_{start} is determined, the wait end time t_{end} is set to the timestamp when the first POB state appears. Finally, the time interval between the wait start time and the end time is added into the taxi wait time set $Y(r)$, i.e., $\{t^m_{wait} | m = 1, 2, \cdots\}$.

5.2 Time Slot with Pickup Event Feature

We divide the time domain into L continuous time slots, where time slot T^j $(1 \leq j \leq L)$ starts at t^{j-1} and ends at

Algorithm 2 Wait Time Extraction Algorithm

Input: $W(r)$ for queue spot r.
Output: Taxi Wait time set $Y(r)$.

$t_{start} \leftarrow null$; $t_{end} \leftarrow null$;
for each sub-trajectory $R(s, e)$ in $W(r)$ **do**
  for $i = s_m \rightarrow e_m$ **do**
    if $p_{i.state} = \{FREE\ or\ ONCALL\ or\ ARRIVED\}$ and $t_{start} = null$ **then**
      $t_{start} \leftarrow p_{i.ts}$;
    else if $p_{i.state} = PAYMENT$ and $t_{start} \neq null$ **then**
      $t_{start} \leftarrow null$; $t_{end} \leftarrow null$;
    else if $p_{i.state} = POB$ and $t_{start} \neq null$ and $t_{end} = null$ **then**
      $t_{end} \leftarrow p_{i.ts}$;
    end if
  end for
  if $t_{start} \neq null$ and $t_{end} \neq null$ **then**
    $Y(r).Add(t_{end} - t_{start})$
  end if
  $t_{start} \leftarrow null$; $t_{end} \leftarrow null$;
end for

t^j. The time t^0 and t^j are the start time and end time of the time domain. Accordingly, set $Y(r)$ can be divided into L partitions, where $Y(r)^j = \{t^m_{wait} | t^{j-1} \leq t^m_{start} < t^j\}$ and $1 \leq j \leq L$.

Given $Y(r)^j$ consists of multiple t^m_{wait}, we have their arithmetic mean, denoted by $\bar{t}_{wait}(r)^j$, as the average taxi wait time over time slot T^j. When computing $\bar{t}_{wait}(r)^j$, we only consider all street jobs' wait time, i.e., t^m_{start} set by the timestamp of FREE, as a booking job's wait time mainly depends on a specific booking passenger's individual arrival time.

Given the arrival number of FREE taxi over time slot T^j, denoted by $N_{arr}(r)^j$, and time slot length $t^j - t^{j-1}$, we have the average arrival rate of FREE taxis over time slot T^j: $\bar{\lambda}(r)^j = \frac{N_{arr}(r)^j}{t^{j+1} - t^j}$. According to Little's Law [7], which relates average arrival rate, average wait time and average queue length, we have the average FREE taxi queue length over time slot T^j: $\bar{L}(r)^j = \bar{t}_{wait}(r)^j * \bar{\lambda}(r)^j$.

Meanwhile, the taxi departure interval can be computed by $t^{m+1}_{end} - t^m_{end}$, where t^m_{end} and t^{m+1}_{end} are the consecutive wait end time in $Y(r)^j$. Accordingly, we have the arithmetic mean of all the departure intervals, denoted by $\bar{t}_{dep}(r)^j$ over time slot T^j. When computing $\bar{t}_{dep}(r)^j$, we consider the departure time intervals of all the departed taxis, i.e., both street job ones and booking job ones. Thus $\bar{t}_{dep}(r)^j$ depicts the departure rate of all departed taxis over time slot T^j.

Finally, we have a 5-tuple to depict time slot T^j of spot r: $\varphi(r)^j = \left\langle \bar{t}_{wait}(r)^j, N_{arr}(r)^j, \bar{L}(r)^j, \bar{t}_{dep}(r)^j, N_{dep}(r)^j \right\rangle$, where $\bar{t}_{wait}(r)^j$ and $N_{arr}(r)^j$ depict the FREE taxi arrival activities, $\bar{L}(r)^j$ gives the FREE taxi queue length, $\bar{t}_{dep}(r)^j$ and $N_{dep}(r)^j$ depict all taxis' departure activities. All the 5 variables are pickup event features.

As described earlier, the MDT logging is an event-driven action, where it records the exact moment that the taxi state switch to FREE, ARRIVED, PAYMENT or POB. Therefore, the values in the 5-tuple are highly accurate and valid.

5.3 Queue Context Disambiguation

Given the 5-tuple for each time slot at a queue spot, we propose a conceptually simple and easily implemented algorithm, called queue context disambiguation (QCD) algorithm, to identify the different queue types defined in Table 3. The basic idea behind the QCD algorithm is that: a passenger queue may exist when taxi pickup events frequently and continuously occur; besides, a passenger queue may also exist when a considerably high proportion of booked taxis appear, even though the frequency of taxi pickup events is not high.

The main input of QCD is the derived 5-tuple feature set for all time slots $\Omega(r) = \{\varphi(r)^j | j = 1, 2, \cdots, L\}$, and the output is the labeled time slots, i.e., C_1, C_2, C_3 and C_4. The complete algorithm is shown in Algorithm 3.

Algorithm 3 Queue Context Disambiguation Algorithm

Input: $\Omega(r)$, thresholds η_{wait},η_{dep},τ_{arr},τ_{dep},η_{dur},τ_{ratio}.
Output: Labeled T^j, where $1 \leq j \leq L$.

Routine 1:
for $j = 1 \rightarrow L$ **do**
  if $\bar{L}(r)^j < 1$ **then**
    if $N_{arr}(r)^j \geq \tau_{arr}$ and $\bar{t}_{wait}(r)^j < \eta_{wait}$ **then**
      Label T^j to C_2;
    else if $N_{arr}(r)^j < \tau_{arr}$ and $\bar{t}_{wait}(r)^j \geq \eta_{wait}$ **then**
      Label T^j to C_4;
    end if
  end if
  if $\bar{L}(r)^j \geq 1$ **then**
    if $N_{dep}(r)^j \geq \tau_{dep}$ and $\bar{t}_{dep}(r)^j < \eta_{dep}$ **then**
      Label T^j to C_1;
    else if $N_{dep}(r)^j < \tau_{dep}$ and $\bar{t}_{dep}(r)^j \geq \eta_{dep}$ **then**
      Label T^j to C_3;
    end if
  end if
end for
Routine 2:
for $j = 1 \rightarrow L$ **do**
  if T^j not labeled and $N_{dep}(r)^j * \bar{t}_{dep}(r)^j > \eta_{dur}$ and $\frac{N_{arr}(r)^j}{N_{dep}(r)^j} < \tau_{ratio}$ and $\bar{L}(r)^j \geq 1$ **then**
    Label T^j to C_1;
  else if T^j not labeled and $N_{dep}(r)^j * \bar{t}_{dep}(r)^j > \eta_{dur}$ and $\frac{N_{arr}(r)^j}{N_{dep}(r)^j} < \tau_{ratio}$ and $\bar{L}(r)^j < 1$ **then**
    Label T^j to C_2;
  end if
end for

In general, the proposed QCD algorithm consists of 2 routines to identify the queue type, which use different criteria to label the time slots. In Routine 1, it firstly examines whether a taxi queue exists during the given time slot by checking the queue length value $\bar{L}(r)^j$. When a taxi queue does not exist, i.e., $\bar{L}(r)^j < 1$, it further analyzes the average taxi wait time $\bar{t}_{wait}(r)^j$ and arrival taxi number $N_{arr}(r)^j$: a considerably large taxi arrival number with a small average

taxi wait time value indicates a passenger queue exist, and thus it labels the time slot as C_2; On the other hand, a small taxi arrival number with a considerably large average taxi wait time value indicates no passenger queue exist, and thus it labels the time slot as C_4. In other words, the taxi arrival number together with the average taxi wait time value serves as the indicator of a passenger queue. When a taxi queue exists, i.e., $\bar{L}(r)^j >= 1$, it analyzes the average taxi departure interval $N_{dep}(r)^j$ and departure taxi number$N_{dep}(r)^j$: a large taxi departure number with a notably small average taxi departure interval value indicates a passenger queue exist, and thus it labels the time slot as C_1; On the other hand, a small taxi departure number with a notably large average taxi leave interval value indicates no passenger queue exist, and thus it labels the time slot as C_3. Note that the average taxi wait time value is no longer a good indicator of a passenger queue when a taxi queue exist, i.e., $\bar{L}(r)^j >= 1$, as the queuing delay becomes significant and even a dominant factor; on the other hand, the average taxi departure interval value is not a good indicator of a passenger queue when a taxi queue does not exist, i.e., $\bar{L}(r)^j < 1$, as some taxis may depart with booking passengers, and the corresponding leave intervals are only determined by the booking passengers' arrival time.

In Routine 2, the QCD algorithm continues to label the time slots that can not be identified in Routine 1: it mainly uses the ratio $\frac{N_{arr}(r)^j}{N_{dep}(r)^j}$, i.e., the ratio of the arrival FREE taxi number to the total departure taxi number (street jobs + booking jobs) over the given time slot, to infer whether a passenger queue exist. More specifically, a small value of $\frac{N_{arr}(r)^j}{N_{dep}(r)^j}$ essentially means a large portion of ONCALL taxis depart from this queue spot, which strongly indicates the difficulty to get a FREE taxi at the current spot and time slot. Meanwhile, if the departure events occur over a long time period, i.e., $N_{dep}(r)^j * \bar{t}_{dep}(r)^j > \eta_{dur}$, the QCD algorithm would infer the existence of a passenger queue, and accordingly labels the given time slot to C_1 or C_2 according to the taxi queue length value $\bar{L}(r)^j$. Note that in Singapore, people usually prefer hailing down a *FREE* taxi rather than booking a taxi, as they have to pay a compulsory 3 to 4 Singapore dollars' taxi booking fee, which easily takes up more than 30% of a single-trip taxi fare.

The threshold values used in the proposed QCD algorithm need to be properly set, and different queue spots may have different threshold values: for example, a queue spot located at a hospital might have distinct threshold values from the one in the airport. We will illustrate how to determine these values in section 6.

6. EMPIRICAL EVALUATION

6.1 Queue Spot Detection Experiment

6.1.1 Data Preprocessing

Our entire dataset contains about 15000 taxis' MDT logs, which occupy around 60% of the total taxis in Singapore. In general, the 15000 taxis generate around 12.38 million daily MDT log records, and each MDT generate 848 daily MDT log records. It is natural that such a large MDT log dataset contains some errors, and the main error types are: (1) improper/missing taxi states; (2) record duplication; (3) GPS coordinates outside Singapore or in inaccessible zones.

Firstly, the standard taxi state transition diagram is given in Fig. 3, but the taxi states in some MDT logs appear at improper places. For example, a FREE state is found between the two PAYMENT states in many MDT logs. We found that it is a software bug caused by the clock synchronization between the old version MDT device and the taximeter. Another common issue is some intermediate taxi states, e.g., ARRIVED, NOSHOW or STC, are missing and lost. Two possible reasons are identified: 1) these intermediate states sometimes last only several seconds and then switch to another state before the MDT logging thread tracks them down. 2) logging some intermediate states requires taxi driver manually press some specific buttons on the MDT touch screen, but some drivers omit this step either purposely or accidentally. Secondly, the duplicate records in the MDT logs are mainly caused by the re-transmission of the GPRS messages between MDT and the backend service. Thirdly, the GPS coordinates errors are normally caused by the urban canyon effect [3].

In short, we remove the above described erroneous records from the raw MDT log dataset, which occupy around 2.8% of the total MDT log records.

6.1.2 Experiment Setup and Parameter Selection

We use the daily MDT logs from all 15000 taxis for the queue spot detection. Firstly, we run the proposed PEA algorithm on the 15000 taxis' daily individual trajectories, where 10 km per hour is as the speed threshold η_{sp}, and successfully extract more than 264000 pickup event sub-trajectories for each single day. Each extracted sub-trajectory provides us one central GPS location, and accordingly we have around 264000 GPS locations, i.e., the GPS location set C, for the DBSCAN clustering. Running the DBSCAN clustering algorithm on such a large-size point set is significantly slow due to its $O(n^2)$ complexity. Therefore, we simply divide Singapore into 4 rectangular zones based on their different characteristics, i.e., Central, North, West and East, as illustrated in Fig. 5. The central zone covers Singapore's central business district(CBD) and most of tourist attractions. The other 3 zones are typically residential and industrial areas with a few tourist attractions. Therefore, the GPS location set C is further divided into 4 subsets, and we run the DBSCAN clustering algorithm on each subset respectively.

Moreover, properly choosing the two parameters of DBSCAN, i.e., eps ε_d and min-points p_d, is not a trivial issue: ε_d specifies the maximum radius of the neighborhood and p_d sets the minimum number of points in an eps-neighborhood of the point. An unduly small ε_d or an overly large p_d may lead a large part of the data points cannot be clustered, while an overly large ε_d or an unduly small p_d would merge different clusters into one. Fig. 6 shows the number of the detected queue spot with respect to different ε_d and p_d. We see that small ε_d values (e.g., 10 meters) or large p_d values (e.g., 100 points) result that only a few number of queue spots are detected and many actual ones are neglected. On the other hand, large ε_d values (e.g., 20 meters) or small p_d values (e.g., 25 points) would easily merge adjacent queue spots and meanwhile bring many insignificant queue spots.

Figure 5: Four Rectangular Zones

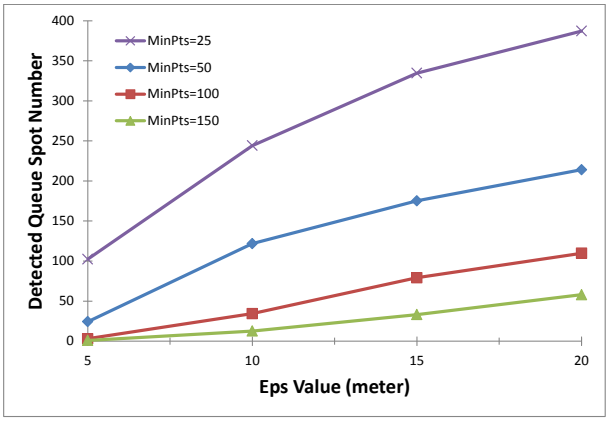

Figure 6: DBSCAN Performance with Different Parameter Pairs

By carefully comparing the DBSCAN clustering results, we finally set its parameter ε_d and p_d to 15 meters and 50 points respectively when processing the daily Singapore taxis' MDT log dataset. Roughly speaking, the selected two parameters allow detect the queue spot that has more than 50 taxi pickup events within its proximity of 15 meters. Note that the selected parameters are used to process the MDT log dataset on the daily basis, and for different time durations, e.g., one week's MDT log dataset, we may have to change and reset the DBSCAN parameters.

6.1.3 Queue Spot Detection Results and Analysis

With the selected DBSCAN parameters, totally around 180 queue spots are detected in the four zones of Singapore, as illustrated in Fig. 7. We analyze and manually label the detected queue spots by locating their GPS coordinates on Google Maps with its Street View feature. It shows that most of the detected queue spots are located nearby the public facilities or landmarks. Table 4 summarizes the result: almost half of the queue spots are nearby Singapore Mass Rapid Transit (MRT) stations or bus stations, around 20% of them are located nearby the shopping malls, hotels or office buildings. Only around 5% of the detected queue spots do not have any significant nearby facility or landmark.

We compare the queue spot detection results with the taxi stand locations, which are sets up by Singapore Land Authority (LTA): there are 31 taxi stands with more than 3 taxi

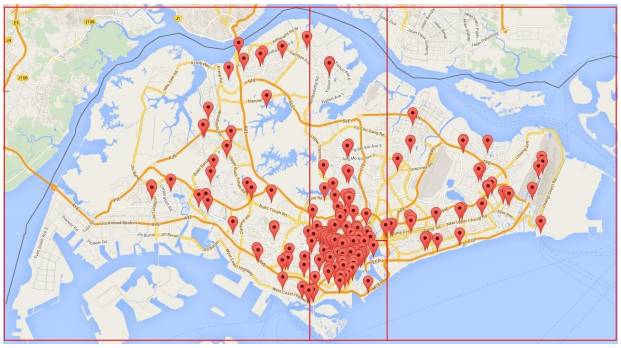

Figure 7: Detected Queue Spots in Singapore

Table 4: Landmark Nearby the Detected Queue Spots

Nearby Facility or Landmark	Detected Spots Percentage
MRT & BUS station	48.3%
Shopping Mall & Hotel	11.8%
Office Building	9.6%
Hospital & School	8.4%
Tourist Attraction	6.2%
Airport & Ferry Terminal	5.6%
Industrial and Residential Area	4.5%
Unidentified	5.6%

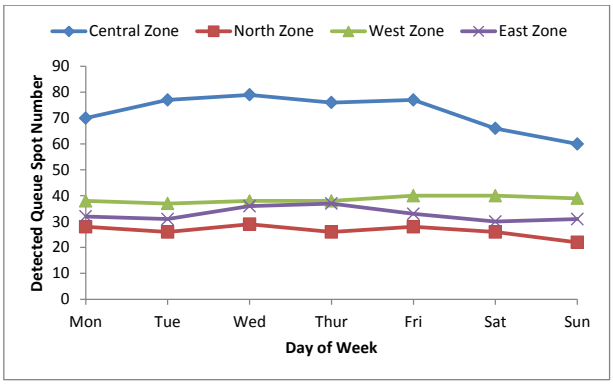

Figure 8: Queue Spot Number in Different Zones and Days

parking lots in the CBD area, and 30 of them are correctly detected with the average location error only 7.6 meters (possibly caused by the GPS error). More importantly, more than 15 queue spots in this area, which are not labeled by LTA, are busy enough and even have more daily pickups than many taxi stands.

We further compare the detected queue spot number in different zones and days of week. Fig. 8 shows that central zone has the largest number of queue spots, although it only occupies around 6% of the total area. The main reason is that most of the high-rise office buildings, shopping malls and tourist attractions in Singapore are located in this zone. More importantly, Fig. 8 shows that queue spot numbers in all zones do not have a high fluctuation on different

Table 5: Hausdorff Distance between Queue Spot Sets on Different Days of Week (Meter)

Hausdorff distance	Mon	Tue	Wed	Thurs	Fri	Sat	Sun
Mon	0	57.076	42.958	59.475	45.531	104.61	143.27
Tue	57.076	0	44.768	34.649	54.293	117.41	141.07
Wed	42.958	44.768	0	41.429	54.311	106.71	139.87
Thurs	59.475	34.649	41.429	0	57.721	111.58	133.21
Fri	45.531	54.293	54.311	57.721	0	81.125	119.41
Sat	104.61	117.41	106.71	111.58	81.125	0	67.111
Sun	143.27	141.07	139.87	133.21	119.41	67.111	0

week days (from Monday to Friday), while the queue spot number slightly drops down on Saturday and Sunday in the central zone. It is probably caused by fewer local working people traveling in the CBD area during the weekend. The queue spot number during weekend does not drop significantly, as still many people go to the shopping malls and tourist attractions in the central zone.

To gain further insight into the queue spot detection results on different days of week, we adopt the modified Hausdorff distance [4] to evaluate their similarity and stability. Hausdorff distance (or called Pompeiu-Hausdorff distance) has been widely used to measure the similarity of two point sets in object matching. Briefly speaking, it is the maximum distance of a point set to the nearest point in the other point set. Therefore, the Hausdorff distance between the detected queue spot sets illustrates whether the queue spot sets on different days of week are stable and how far they are from each other. Table 5 shows the Hausdorff distance between the detected queue spot sets in meter, where each set consists of all the four zones' detected queue spots.

Based on the definition of the Hausdorff distance, the value "0" means the two spot sets are overlapped exactly, and a large distance value means big mismatch between the two queue spot sets. From Table 5, we see that the Hausdorff distances between any two queue spot sets are only around 50 meters on the week days and around 67 meters on the weekend days: it indicates that the detected queue spots match quite well and thus the queue locations are normally stable. When we compare a week day queue spot set with a weekend queue spot set, e.g., Monday and Sunday, the corresponding Hausdorff distance easily increases to around 130 meters, but it is still a relatively low value given Singapore an area with 50 kilometers long and 26 kilometers wide.

In short, the proposed queue spot detection method with the properly selected parameters effectively detects the Singapore island-wide queue spots with a high accuracy and stability.

6.2 Context Disambiguation Experiment

6.2.1 *Feature Preparation and Threshold Selection*

For each detected queue spot, say r, we have its corresponding pickup event sub-trajectory set $W(r)$, which usually consists of a number of sub-trajectories ranging from 100 to 500 daily. Table 6 illustrates the daily average number at a queue spot in different zones and days. We see that a queue spot has around 200 sub-trajectories on average on a week day, and the number slightly goes up on a weekend day. Meanwhile, the average number in the east zone is always higher than the other 3 zones: it is probably caused by a large number of pickup events at Singapore's Changi international airport, which is located at the east zone.

Table 6: Average Pickup Event Number

Avg. Sub-trajectory Number	Central	North	West	East
Working Day	217.5	165.5	223.3	267.2
Weekend Day	251.6	172.3	198.1	305.8

We conduct the experiment on a daily basis and divide one day into 48 fixed-size time slot. Each time slot thus takes 1800 seconds, e.g., 00:00 to 00:30 or 18:30 to 19:00. For each detected queue spot, say r, we run the WTE algorithm on its pickup event sub-trajectory set $W(r)$, and accordingly derive the 5-tuple feature set for all 48 time slots, i.e., $\Omega(r) = \{\varphi(r)^j | j = 1, 2, \cdots, 48\}$.

Before running the QCD algorithm on $\Omega(r)$, we need to determine the 6 threshold values used in the algorithm. For each queue spot, we select its top 20% shortest wait time values and top 20% shortest departure intervals, which can commonly depict taxi wait and departure events when the passenger queue exists. We thus use their average values as the threshold η_{wait} and η_{dep} respectively. Accordingly, we set the threshold τ_{arr} and τ_{dep} to $\frac{1800}{\eta_{wait}}$ and $\frac{1800}{\eta_{dep}}$ respectively, where 1800 is the predetermined time slot length in seconds. Meanwhile, the threshold η_{dur} is set to 90% of the current time slot length, namely 1620 seconds. To determine the threshold τ_{ratio}, we calculate the daily ratio of the total street job number to the total job number (street jobs + booking jobs) in different zones and days of week, and then set the threshold τ_{ratio} to the corresponding ratio value, e.g., 0.84 is the average ratio value in the central zone on Sunday. The taxi state transition knowledge, as illustrated in Fig. 3, is directly used to derive and separate booking jobs and street jobs from the MDT logs.

Lastly, given the fact that our dataset, i.e., around 15000 taxis' MDT logs, only occupies 60% of the total operating

Table 7: Proportion of Different Queue Types

Queue Type	Percentage in All Time Slots
C_1	30.1%
C_2	11.7%
C_3	8.6%
C_4	33.1%
Unidentified	16.5%

taxis in Singapore, we increase the feature values $N_{arr}(r)^j$, $\bar{L}(r)^j$ and $N_{dep}(r)^j$ by multiplying an amplification factor 1.667 and decrease the feature value $\bar{t}_{dep}(r)^j$ by multiplying 0.6 for all time slots.

6.2.2 *Experiment Results and Analysis*

Based on the extracted 5-tuple feature set and the determined threshold values, we run the QCD algorithm on 25 randomly selected queue spots respectively. Table 7 summarizes the queue type identification results: nearly 84% time slots in total are successfully labeled as C_1 to C_4 respectively, among which 30% time slots are labeled as C_1, namely taxi queue and passenger queue concurrently exist, while 33% time slots are labeled as C_4, namely no any taxi queue or passenger queue during these time slots. Meanwhile, only 11.7% time slots are labeled as C_2, namely only passenger queue exists, and 8.6% time slots are labeled as C_3, namely only taxi queue exists. It shows that such two situations do not occur as frequently as C_1 or C_4. Besides, around 16% time slots cannot be identified by the QCD algorithm due to their insignificant features. For example, during a time slot T^j that no taxi queue exist, i.e., queue length $\bar{L}(r)^j < 1$, only several taxis, e.g., 7 or 8, arrive and depart with a moderate average wait time value; meanwhile, there is no significant number of *ONCALL* taxis arrive and leave. In such cases, the QCD algorithm does not label the time slot to any predefined type, but simply label it as unidentified or insignificant type.

We further compare the queue type identification results on different days of week. Fig. 9 shows that during the five week days, all queue types' proportion do not have a high fluctuation. However, during the two weekend days, especially on Sunday, the proportion of C_4 significantly goes up from 30% to around 40%, and meanwhile the proportions of C_2 and the unidentified time slots drop down. One possible explanation is that during weekend days fewer business and working people traveling leads to more C_4 time slots and accordingly fewer C_2 and unidentified time slots. Fig. 9 also shows that the proportion of C_3 slightly goes down and C_1 generally maintains its proportion over a whole week.

To further validate the queue type identification results, we collect the waiting taxi number from an independent vehicle monitor system [14], which is set up for continuously observing the vehicle number inside a taxi stand area (normally a predefined polygon). The monitor system updates the vehicle number every 60 seconds via its RESTful web service, which can be used as a good indicator for the existence of a taxi queue. On the other hand, we take the failed taxi booking records from the taxi operator's backend database.

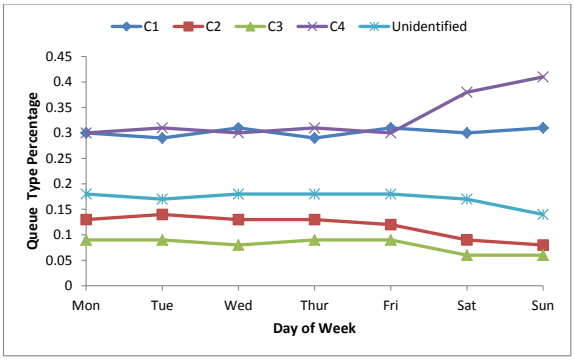

Figure 9: Proportion of Queue Type in Different Days of Week

Table 8: Average Number of Taxis and Failed Bookings

Queue Type	Avg. Taxi Number	Avg. Failed Booking Number
C_1	6.13	0.35
C_2	1.35	4.29
C_3	3.26	0.13
C_4	0.32	0.73
Unidentified	1.56	0.24

A failed taxi booking means the booking request has been successfully dispatched to all the nearby taxis, but the passenger finally fails to get a taxi due to no taxi available inside the dispatching circle centered at the pickup location with radius 1 kilometer. Frequently failed bookings over a short period at the same pickup location indicate that the passenger's current demand is much higher than the taxi's current supply, and thus can be used to imply the existence of a passenger queue.

Table 8 shows the average taxi number obtained from the vehicle monitor system and the average failed bookings number obtained from the taxi operator during the labeled time slots. We see that the average taxi numbers of C_1 and C_3 are notably higher than the corresponding values of C_2 and C_4: it suggests a high chance of a taxi queue occurring during the time slots labeled as C_1 and C_3. Meanwhile, the average failed booking numbers of C_2 is significantly higher than the others: it strongly indicates a high chance of a passenger queue occurring during the time slots labeled as C_2. Note that the time slots labeled as C_1 do not have many failed bookings: it is probably because many available taxis queuing at or nearby the queue spot. In short, the failed booking data combined with the information from an independent vehicle monitor system, at least to some extend, validates the queue type identification results.

6.2.3 *A Sample Case: Lucky Plaza Queue Spot*

To demonstrate the queue type identification results from the individual perspective, we take one queue spot, which is detected nearby the main entrance of Singapore Lucky Plaza, as an illustrative example. Lucky Plaza is a shopping center located at Orchard Road, which is a famous retail and entertainment hub in Singapore. We simply pick up a Sunday's queue type identification results at the Lucky

Table 9: A Sample Queue Type Identification Result

Queue Type	C1	C2	C3	C4	Unidentified
Time Slot	00:00 --- 00:30 09:30 --- 10:00 11:00 --- 15:30 17:30 ---19:30	15:30 --- 17:30 19:30 --- 20:00	00:30 --- 1:30 10:00 --- 11:00 20:00 --- 21:30	1:30 --- 08:30 21:30 --- 23:30	8:30 --- 9:30 23:30 --- 24:00

Plaza queue spot, and summarize them in Table 9.

From Table 9, we see that during the early midnight the queue type C_1 (from 00:00 to 00:30) and C_3 (from 00:30 to 01:30) are identified, which means the concurrent passenger queue and taxi queue occur first and then only the taxi queue left. After that, the queue type C_4 lasts 7 hours (from 01:30 to 8:30), meaning no taxi queue or passenger queue until the early morning. During the peak shopping hours (from 11:00 to 20:00), the queue type shifts between C_1 and C_2, meaning either the concurrent passenger queue and taxi queue or only the passenger queue at Lucky Plaza. After the peak shopping hours, the queue type switches back to C_4 (from 21:30 to 23:30), i.e., no taxi queue or passenger queue. We conducted a short term study at the Lucky Plaza queue spot: the actual queue variance pattern fits well with the above described queue type identification result. For example, during the early midnight, most of people, who just leave the nearby night clubs, usually wait taxis at this queue spot, which explains why the queue type changes to C_1 or C_3 during such time slots.

7. DISCUSSION

7.1 A Real World Deployment

To better serve the stakeholders of our solution (the government agencies, public commuters and taxi operator), we implemented a practical taxi queue detection and analysis system, which consists of a backend queue analytic engine and a web-based frontend user interface.

Within the queue analytic engine, the queue spot detection module uses the relatively long-term historical dataset to extract queue spots. Based on the evaluation results, the queue spot sets in Singapore show some differences between week days and weekend days. Thus, we simply use historical week days' dataset to extract queue spots for a week day, and use historical weekend days' dataset to extract queue spots for a weekend day. In the current implementation, the queue spot detection module collects the most recent 5 week days' dataset and 2 weekend days' dataset to extract and update the corresponding queue locations. The queue context disambiguation module currently mainly runs on the short-term historical dataset to generate the queue type transition reports for week day and weekend day respectively. The historical dataset size and the update frequency for both modules can be configured by system users.

The web-based user interface provide users a simple way to understand and access the queue detection and analysis results. When a user opens the query page, it firstly shows all the latest detected queue spots in Singapore on the Google Map. By zooming into a region and placing the mouse on a specific queue spot, the user can see the identified queue type at the selected time slot together with the nearby facility name as shown in Fig. 10. If necessary, the user can

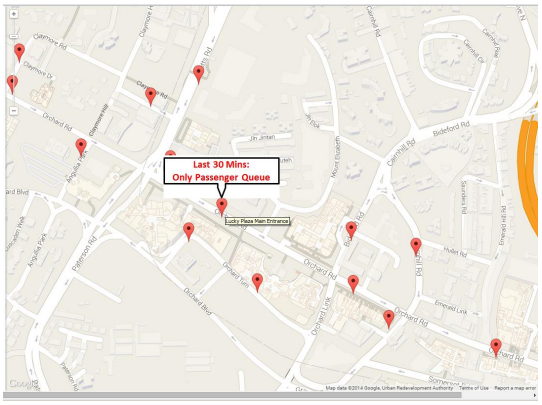

Figure 10: User Interface of the Deployed System

further query the long-term queue type transition reports and save it into the database or a text file.

The backend queue analytic engine is mainly implemented in Java and uses Java database connectivity (JDBC) to retrieve the readily available MDT logs in a PostgreSQL database system. The frontend user interface is developed based on Google Map Javascript API.

7.2 Interesting Findings

Driver Behavior: the taxi *BUSY* state, as describe in Table 1, is designed for taxi drivers when they temporarily unavailable due to personal reasons. However, we find that during the time slots of C_1 and C_2, especially C_2 (namely only passenger queue), a number of taxis enter the queue spots with a *BUSY* state and then quickly leave with a *POB* state. Such a phenomenon indicates that some taxi drivers only pick up their favorite passengers and deny the others by using the *BUSY* state as an excuse. We are currently further investigating on this issue and the possible solutions.

Sporadic Queue Spot: although the detected queue spots in Singapore show an overall high stability, there are a few exceptions. For example, a queue spot inside the west zone periodically appears only on every Sunday (occasionally on Saturday) but never shows during week days. By manually labeling this queue spot, we locate it at a local leisure park, which is not a quite famous tourist attraction in Singapore but only a popular place for the local family during the weekend. Another example is that an individual queue spot is detected only on a specific Sunday at Sentosa island, and through the local newspaper we find that it is caused by a company's anniversary celebration event. These examples not only show that the sporadic queue spots can be precisely captured by our solution, but also demonstrate that meaningful semantics behind both regular and sporadic queue spots can be further explored.

8. RELATED WORK

8.1 Queue Sensing and Detection

Queuing theory [6] has been extensively studied and widely applied (e.g., in the wireless communications), while there is limited research work on detection and analysis of the real world queuing behaviors. The early studies [10] conduc-

t video analytics to detect human queuing activity based on stationary cameras. Recently, people start using smartphones [12] and sensor networks [11] to sense and detect human and vehicle queues. All the existing solutions require fixed infrastructures with maintenance overheads and additional costs.

8.2 Taxi Trace Analytics

Driven by the availability of abundant information in taxi traces, especially the GPS location information, mining taxi traces has received massive attentions from both academia and industry in recent years. The existing work can be generally classified into three categories: 1) mining taxi traces to study the city population movement patterns and behaviors [9, 15]; 2) using taxi traces as a probe to infer or predict traffic conditions for city road networks [1, 8]; 3) mining taxi traces to discover and sense human or vehicle's special events and behaviors [16, 17]. For example, the authors in [16] use taxi traces to sense refueling behavior and citywide petrol consumption. Our work can be classified into the third category, and we refer the interested readers to a good survey [2] for taxi traces analytics.

To the best of our knowledge, there is no previous work using taxi traces to conduct the real world queue analytics for both taxis and their passengers.

9. CONCLUSION AND FUTURE WORK

By leveraging on the event-driven MDT logs and the taxi state transitions, we design and implement a practical system to effectively detect the queue spots and identify the queue context in Singapore. The extensive evaluation results show the feasibility and the accuracy of the deployed queue analytic engine.

The deployed system and the queue context analysis results lay a solid foundation for future work:

- Integrate the queue analytic information into the existing MDT system to conduct recommendations for taxi drivers, e.g., suggesting recent emerging passenger queue spots.
- Periodically publish both taxi queue and passenger queue information to the public and help to reduce queuing events in the city.
- Work with LTA to set up new taxi stands at the busy queuing spots and improving the existing facilities.

Lastly, we would like to highlight that the nature of this work is applicable to not only Singapore but also other densely populated cities, given their taxis equipped with the MDT-like telematics devices.

10. REFERENCES

[1] J. Aslam, S. Lim, X. Pan, and D. Rus. City-scale traffic estimation from a roving sensor network. In *Proc. ACM Conference on Embedded Network Sensor Systems (SenSys)*, 2012.

[2] P. S. Castro, D. Zhang, C. Chen, S. Li, and G. Pan. From taxi gps traces to social and community dynamics: A survey. *ACM Comput. Surv.*, 46(2):17:1–17:34, Dec. 2013.

[3] M. Dottling, F. Kuchen, and W. Wiesbeck. Deterministic modeling of the street canyon effect in urban micro and pico cells. In *Proc. IEEE International Conference on Communications (ICC)*, pages 36–40, June 1997.

[4] M.-P. Dubuisson and A. Jain. A Modified Hausdorff Distance for Object Matching. In *Proc. IAPR International Conference on Computer Vision and Image Processing*, pages 566–568, Oct. 1994.

[5] M. Ester, H. peter Kriegel, J. S, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 1996.

[6] D. Gross, J. F. Shortle, J. M. Thompson, and C. M. Harris. *Fundamentals of Queueing Theory*. Wiley-Interscience, New York, NY, USA, 2008.

[7] J. D. C. Little. A Proof for the Queuing Formula: L = λW. *Operations Research*, 9(3):383–387, 1961.

[8] S. Liu, Y. Liu, L. M. Ni, J. Fan, and M. Li. Towards mobility-based clustering. In *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2010.

[9] L. Moreira-Matias, J. Gama, M. Ferreira, J. Mendes-Moreira, and L. Damas. Predicting taxi-passenger demand using streaming data. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1393–1402, Sept 2013.

[10] J. Segen. A camera-based system for tracking people in real time. In *Proc. International Conference on Pattern Recognition (ICPR)*, 1996.

[11] R. Sen, A. Maurya, B. Raman, R. Mehta, R. Kalyanaraman, N. Vankadhara, S. Roy, and P. Sharma. Kyun queue: A sensor network system to monitor road traffic queues. In *Proc. ACM Conference on Embedded Network Sensor Systems(SenSys)*, 2012.

[12] Y. Wang, J. Yang, Y. Chen, H. Liu, M. Gruteser, and R. P. Martin. Tracking human queues using single-point signal monitoring. In *Proc. ACM Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2014.

[13] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. San Francisco: Morgan Kaufmann, 2005.

[14] W. Wu, W. S. Ng, S. Krishnaswamy, and A. Sinha. To taxi or not to taxi? - enabling personalised and real-time transportation decisions for mobile users. In *Proc. IEEE International Conference on Mobile Data Management (MDM)*, pages 320–323, July 2012.

[15] N. J. Yuan, Y. Zheng, L. Zhang, and X. Xie. T-finder: A recommender system for finding passengers and vacant taxis. *IEEE Transactions on Knowledge and Data Engineering*, 25(10):2390–2403, 2013.

[16] F. Zhang, D. Wilkie, Y. Zheng, and X. Xie. Sensing the pulse of urban refueling behavior. In *Proc. ACM International Joint Conference on Pervasive and Ubiquitous Computing(UbiComp)*, 2013.

[17] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma. Mining interesting locations and travel sequences from gps trajectories. In *Proc. ACM Conference on World Wide Web (WWW)*, New York, NY, USA, 2009.