A modified Density-Based Scanning Algorithm with Noise for spatial travel pattern analysis from Smart Card AFC data

Article *in* Transportation Research Part C Emerging Technologies · March 2015

DOI: 10.1016/j.trc.2015.03.033

CITATIONS
35

READS
1,002

3 authors:

Minh Le Kieu
University of Auckland
28 PUBLICATIONS **268** CITATIONS

SEE PROFILE

Ashish Bhaskar
Queensland University of Technology
110 PUBLICATIONS **884** CITATIONS

SEE PROFILE

Edward Chung
The Hong Kong Polytechnic University
196 PUBLICATIONS **1,648** CITATIONS

SEE PROFILE

Article:

Le-Minh Kieu, Ashish Bhaskar, Edward Chung, A modified Density-Based Scanning Algorithm with Noise for spatial travel pattern analysis from Smart Card AFC data, Transportation Research Part C: Emerging Technologies, Available online 6 April 2015, ISSN 0968-090X, http://dx.doi.org/10.1016/j.trc.2015.03.033.

Keywords: Spatial travel pattern; Public transport; Smart Card; AFC; DBSCAN

A Modified Density-Based Scanning Algorithm with Noise for spatial travel pattern analysis from Smart Card AFC data

Le-Minh Kieu (Corresponding Author)
Smart Transport Research Centre
School of Civil Engineering and Build Environment, Science and Engineering Faculty
Queensland University of Technology
Brisbane, Australia.
Email: leminh.kieu@student.qut.edu.au

Ashish Bhaskar
Smart Transport Research Centre
School of Civil Engineering and Build Environment, Science and Engineering Faculty
Queensland University of Technology
Brisbane, Australia.
Email: ashish.bhaskar@qut.edu.au

Edward Chung
Smart Transport Research Centre
School of Civil Engineering and Build Environment, Science and Engineering Faculty
Queensland University of Technology
Brisbane, Australia.
Email: edward.chung@qut.edu.au

Abstract

Smart Card Automated Fare Collection (AFC) data has been extensively exploited to understand passenger behavior, passenger segment, trip purpose and improve transit planning through spatial travel pattern analysis. The literature has been evolving from simple to more sophisticated methods such as from aggregated to individual travel pattern analysis, and from stop-to-stop to flexible stop aggregation. However, the issue of high computing complexity has limited these methods in practical applications. This paper proposes a new algorithm named Weighted Stop Density Based Scanning Algorithm with Noise (WS-DBSCAN) based on the classical Density Based Scanning Algorithm with Noise (DBSCAN) algorithm to detect and update the daily changes in travel pattern. WS-DBSCAN converts the classical quadratic computation complexity DBSCAN to a problem of sub-quadratic complexity. The numerical experiment using the real AFC data in South East Queensland, Australia shows that the algorithm costs only 0.45% in computation time compared to the classical DBSCAN, but provides the same clustering results.

Introduction

Smart Card Automated Fare Collection (AFC) system has been increasingly popular in public transport, providing a massive quantity of continuous and dynamic data on passenger boarding and alighting locations. This information provides a tremendous opportunity to analyze spatial travel patterns of the transit users-defined in terms of the regular boarding and alighting stops of transit passengers. An emerging number of studies have extensively explored multiday travel pattern (Chu and Chapleau, 2010; Kieu et al., 2014; Ma et al., 2013) to understand individual travel behaviors (Ma et al., 2013), passenger segmentation (Kieu et al., 2014), trip purpose (Lee and Hickman, 2014) and potential in transit planning (Utsunomiya et al., 2006). A detailed review of existing advances in AFC analysis could be found in Pelletier et al. (2011).

Spatial travel pattern is defined in this paper as regular origin-destination (OD) that transit passenger usually travels between. The literature of travel pattern analysis using AFC data has been evolving from simple to more sophisticated methods such as from aggregated to individual travel pattern analysis, and from stop-to-stop to flexible stop aggregation. Although considerable research has recently been devoted to capturing the individuality and travel behaviors of transit passengers, rather less attention has been paid to the practical computing constraints which limit those methods in real-world applications. So far, existing method have been confined to first-time analysis of spatial travel pattern – i.e. finding the travel pattern from AFC data without any prior knowledge of passenger travel pattern, leaving the question of updating this information from the existing travel pattern knowledge unanswered. To make the best use of the individual travel pattern in customized service provision and operational strategies, travel pattern has to be updated daily to observe the changes in passenger behaviors. After the last service of the day, transit operator collects all Smart Card transactions of the day and updates travel pattern of each

individual passenger before the first service of the next day. However, existing methods have been developed with increasing complexity and degree of detail, while the number of Smart Cards and transit journeys are also growing rapidly. A full spatial travel pattern analysis would be an absurd task to perform within this short time gap of a few minutes to hours. Consequently, there is a need for a time-effective algorithm to observe and record the evolution of travel pattern.

This paper proposes a new algorithm named Weighted Stop Density Based Scanning Algorithm with Noise (WS-DBSCAN) based on the classical Density Based Scanning Algorithm with Noise (DBSCAN) aiming to rapidly detect and update individual spatial travel patterns while maintains high degree of detail in travel pattern analysis, which enables transit operators to use this information on a daily basis. This research focuses on spatial travel pattern analysis only, because spatial pattern analysis as a two-dimensional problem has much more complexity than temporal travel pattern. We also focus on developing a new algorithm to detect and update individual travel pattern, without actual analysis to mine spatial and temporal travel patterns from transit passengers.

The paper first reviews the existing studies on travel pattern analysis using AFC data. After the description of the data processing method, the paper describes the DBSCAN implementation in travel pattern analysis. The WS-DBSCAN algorithm along with an example of its implementation is then presented. The numerical experiment shows the effectiveness of the proposed method compared to the classical DBSCAN algorithm. A discussion and future research directions finally concludes the paper.

Literature review

The use of AFC data enables us to continuously analyze the multiday travel patterns of a much larger population than the traditional travel survey method. The existing studies in the literature have explored travel pattern by different level of aggregation on passenger and stop level.

Existing passenger aggregation approaches for travel pattern analysis

An emerging number of publications have recently analyzed transit passenger travel pattern by different level of aggregation from whole aggregated dataset to each individual Smart Card user. Utsunomiya et al. (2006) is an example of aggregated dataset analysis. The authors described the data possessing and analysis methods to mine meaningful information from AFC data. Jang (2010) demonstrated the use of AFC data in travel time and transfer locations analysis. The method facilitates the comparison between different transit modes and the identification of passenger transfer choices. Hasan et al. (2013) exploited AFC data to observe both spatial and temporal passenger travel pattern. The authors modeled two important passenger decisions: (a) which place to visit (by assuming a fixed probability of visit to each regular place) and (b) how

long to stay (by a hazard based duration modeling). The whole dataset analysis explores general travel patterns from transit passengers.

Some other authors emphasized the similarity of travel pattern by subgroup. Their analyses are based on the aggregation of several similar characteristics of the transit trip and passenger. Morency et al. (2007) aggregated the Smart Card users into five classes according to the card type and the privilege of route usage. The travel profile of each card type could be well observed by investigating the indicators of spatial and temporal travel pattern. Chu et al. (2009) proposed a new framework to mine spatial-temporal distribution of transit demand by different aggregation level such as stop, route, link, node and card type. Lee and Hickman (2014) developed a heuristic rules algorithm and a classification decision tree to group Smart Card users into multiple classes and infer their trip purposes.

Although aggregated travel pattern analysis provides insights into the travel pattern of general user, it fails to capture the individuality of travel behavior. The typologies passenger groups are also predefined which might not reflect the similarity of passengers between the same class, and the difference between classes.

Several studies have recently enriched the travel pattern comprehension by individually analyzing each Smart Card user. Chu & Chapleau (2010) described a disaggregated travel pattern analysis framework for multi-day AFC data. "Anchor points" or repeated travel locations are mined from each Smart Card user and then assigned to known spatial coordinates. Ma et al., (2013) and Kieu et al., (2014) used the classical DBSCAN algorithm, originally proposed in Ester et al. (1996), to mine spatial and temporal travel patterns from AFC data.

Existing stop aggregation approaches for travel pattern analysis

Travel pattern analysis often spatially breaks down to stop-to-stop repeated journeys. However, the limitation of this method has been identified by several authors (Lee and Hickman, 2013). A transit stop is usually linked with only a single direction or route, while transit passengers normally have several route choices options within their origin destination locations. Any stops within the immediate vicinity that provide the same access should be considered in the same travel pattern, because transit passengers might choose them randomly or by the first arriving schedule. In literature different stop aggregation approaches are proposed to group spatially close stops into the same travel pattern. Chu & Chapleau (2010) aggregated stops within 50m of each other to form a new node. Lee et al. (2012) and Lee & Hickman (2013) proposed a model named "Stop aggregation model" to group stops according to the proximity, stop description and catchment area. Ma et al. (2013) and Kieu et al. (2014) applied the DBSCAN algorithm for stop aggregation model. Compare to other approaches, DBSCAN provides flexibility in defining the group of stops that the passenger repeatedly choose. DBSCAN clusters stops of close proximity, and the travel pattern is defined according to the number of repeated journeys.

Table 1 summarizes the aforementioned review of the existing advances in travel pattern analysis using AFC data.

Table 1 Comparative overview of the literature on travel pattern analysis using Smart Card data

Paper	SC data type	Passenger aggregation	Stop aggregation	Method	Aim
Utsunomiya et al. (2006)	Entry only	Dataset	No	Statistic	Investigate the potential of AFC in transit planning
Jang (2010)	Entry-Exit	Dataset	No	Statistic	Travel time and transfer analysis
Hasan et al. (2013)	Entry only	Dataset	No	Simulation	Spatial-temporal analysis
Morency et al. (2007)	Entry only	Group	No	Data mining	Spatial-temporal transit use variability
Chu et al. (2009)	Entry only	Group	No	Data mining	Passive survey from AFC data
Lee & Hickman (2014)	Entry only	Group	Yes	Rules-based heuristic and data mining	Trip purpose inference
Chu & Chapleau (2010)	Entry only	Individual	Yes	Data mining	Multiday spatial-temporal analysis
Ma et al. (2013)	Entry only	Individual	Yes	Data mining	Multiday spatial-temporal analysis
Kieu et al. (2014)	Entry-Exit	Individual	Yes	Data mining	Passenger segmentation

Methodology

Existing studies provide insights into the problem of travel pattern analysis. After the description of the dataset and itineraries reconstruction process, this section presents the classical DBSCAN application to travel pattern analysis and proposes the WS-DBSCAN algorithm.

Data description

The AFC data used in this study comes from Translink, the transit authority of South East Queensland (SEQ), Australia. The dataset is a compilation of approximately 34.8 million transactions made by a million Smart Cards over 15 thousands transit stops of the bus, city train and ferry networks in SEQ from 1st March to 30th June 2012. Each transaction contains the following fields:

1) *CardID*: Unique Smart Card ID
2) *T_on*: Timestamp for touch on (boarding)
3) *T_off*: Time stamp for touch off (alighting)

4) *S_on*: Station ID at touch on
5) *S_off*: Station ID at touch off
6) *ValidIndicator*: A binary indicator for differentiating valid from invalid transactions. It has been used by the operator for ticketing purpose. Valid transaction is the combination of a touch on and a touch off from the same transit line, within a 2 hours limit (Translink, 2007). Any cases other than that, e.g. no touch off, or touch off at a different line, etc. are indicated as invalid transactions. Only around 3% of the transactions are invalid.
7) *RouteUsed*: The transit line that the passenger has used.
8) *Direction*: Direction of travel (Inbound/Outbound)

For the current analysis, the study is performed only on working days (weekdays excluding public holidays and school holidays) because travel behavior on working weekdays can be significantly different than that of weekends and holidays.

Reconstruction of travel itineraries

The first step in travel pattern analysis is to reconstruct the full travel itinerary from individual transactions. The flowchart on Figure 1 illustrates the algorithm to connect individual transactions from each SC user on each working day into completed journeys from first origin stop to the last alighting stop. The algorithm is built on a binary "*ReconstructingIndicator*" to identify on-going/new journey status; and a "*JourneyID*" to distinguish the completed journeys.

A fixed threshold of 60 minutes is then used to decide if the two transactions are connected. This threshold has been chosen differently in the literature (Seaborn et al., 2009). The 60 minutes is chosen in accordance with Brisbane's public transport threshold for continuation journeys (Translink, 2007). Here, the first origin stop and the last alighting stop of a completed journey are defined as the *"origin stop"* and the *"destination stop"*, respectively. The following 4 steps describe the journey reconstruction process.

1) STEP 1: A binary *ReconstructingIndicator* is defined and assigned as zero.
2) STEP 2: The *ValidIndicator* is checked. If the indicator is equal to 0 (which denotes an invalid transaction) the corresponding journey will be discarded.
3) STEP 3: If *ReconstructingIndicator* is zero, a variable *OriginLocation* is defined and set as equal to the current *T_on*. We also assign a new unique *JourneyID*, change *ReconstructingIndicator* to one, save the current transaction and move to the next transaction.
   If *ReconstructingIndicator* is one and the time gap between the current *T_on* and the last *T_off* is less than 60 minutes, we move to Step 4.
   If the time gap is more than 60 minutes, transactions with previous *JourneyID* is connected into a completed journey. New *JourneyID* and *OriginLocation* are assigned, in which the current transaction is identified as the first leg from the origin stop. The *ReconstructingIndicator* is set as 1.

4) STEP 4: If the current *S_off* is different to *OriginLocation*, the transaction is connected to the journey as a continuation journey and we move to the next transaction. If it is the last transaction of the day, or *S_off* is equal to *OriginLocation*, the journey reconstruction process is completed and we move back to Step 1.

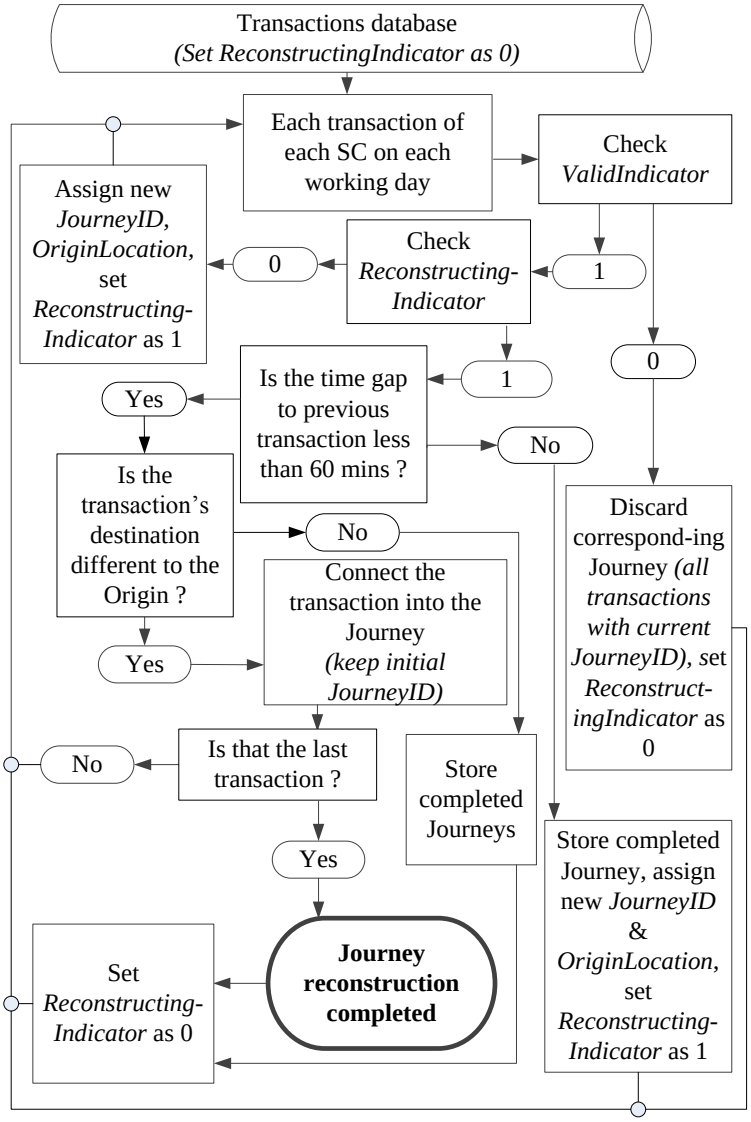


Figure 1 Journey reconstruction flowchart

The four-step process reconstructs full itineraries from AFC data, enables us to observe the origin-destination locations and journey chains of each individual passenger. **Table 2** shows an example of reconstructed itineraries from two transit passengers in 1st Apr 2012.

Table 2 Example of travel itineraries of two random passengers

SC ID	Day	Journey ID	Origin Time (min from 0h)	Destination Time (min from 0h)	Stop ID Sequence	Total travel time (min)	Total transfer time (min)	Total time (min)	Route ID Sequence
X1	1 Apr	1697	680.2	686.54	5→4	6.34	0	6.34	999
X1	1 Apr	1083	557	566.9	54371→24653	9.9	0	9.9	726
X2	1 Apr	1415	898.08	905.24	12861→2452→15212	7.16	41.42	48.58	550→562
X2	1 Apr	1412	887.45	944.63	10730→499→88	57.18	32.7	89.88	690→999

The classical DBSCAN algorithm

The existing literature promotes DBSCAN as one of the best solutions for spatial travel pattern analysis of individual passenger for the following reasons (Kieu et al., 2014; Ma et al., 2013)

1) As mentioned in the Literature review, DBSCAN provides a flexible solution to spatial travel pattern analysis. DBSCAN identifies clusters of high density and also noise which does not belong to any clusters. In travel pattern analysis, noise is anomaly pattern and clusters are the regular spatial travel pattern.
2) DBSCAN identifies cluster of any shape and sizes. In spatial travel pattern analysis, the clustered transit stops could form any shape and sizes.
3) DBSCAN does not require predetermination of initial cores or number of clusters. This feature is also very important for travel pattern analysis because the number of patterns from an individual passenger is unknown.

This section first describes the classical DBSCAN algorithm and thereafter proposes a modified DBSCAN algorithm to dynamically identify individual travel pattern.

The classical DBSCAN algorithm defines clusters as dense regions, separated by regions of lower density. The algorithm has two global parameters: the maximum density reach distance ε and the minimum number of points *MinPts*. We define the ε*-neighborhood* for a point *p* as the number of points in the dataset that has distance to *p* less than ε, including the point *p* itself. The most common distance metric used is the Euclidean distance. Each point in the data set is classified as:

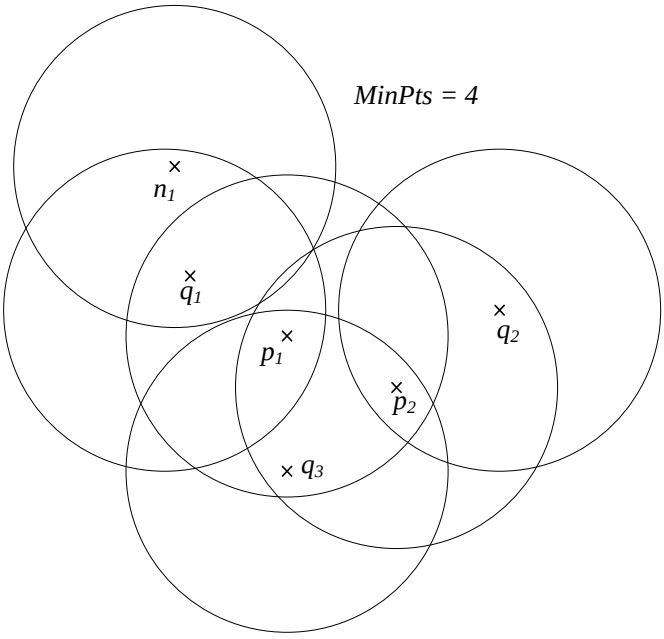
- *Core point*: A point is considered as a *core point* if its ε*-neighborhood* is greater than or equal to *MinPts*.
- *Border point*: A point is considered as a *border point* if its ε*-neighborhood* is less than *MinPts* but the point in itself lies within ε*-neighborhood* of a *core point*.
- *Noise*: A point is considered as a *noise* if it is neither a *core* nor a *border* point.

A cluster is defined by combining the *core* points which are connected by their associated *border points*. Interested reader can refer to Ester et al. (1996) for more detailed description of DBSCAN. Figure 2 provides an example of *core, border point* and *noise* in DBSCAN definitions. The illustrated circles are centered at a point and have a radius of ε.Considering *MinPts* as 4:

a) Point p_1 is a *core point* because its ε *-neighborhood* is four (q_1,q_3,p_2 and p_1 itself). It's a *core point* because this ε *-neighborhood* is equal to *MinPts*. Similarly point p_2 is also a *core point*.

b) Point q_1 is a *border point* because its ε *-neighborhood* is 3 (p_1,n_1 and q_1 itself) and point q_1 lies within the ε *-neighborhood* of the core point p_1. Similarly point q_2 and q_3 are *border points*.

c) Point n_1 is the noise because its ε *-neighborhood* is only 2 and it is not within the ε *-neighborhood* of any *core point*.



The circles show the maximum density reach distance ε
p_1,p_2 are core points; q_1,q_2,q_3 are border points
and n_1 is a noise

Figure 2 An example to illustrate *core point, border point and noise point* definitions for DBSCAN

Kieu et al. (2014) proposed a two-level spatial travel pattern mining procedure from reconstructed itineraries using the classical DBSCAN algorithm. Figure 3 illustrates this spatial travel pattern analysis for a random passenger. Here, the O points represent the origin stops and D points represent the destination stops in the historical itineraries. The two levels are described as follows

1) *Level 1:* DBSCAN algorithm clusters origin stops into regular origin stops and

anomaly origin stops. The DBSCAN algorithm with ε equals 1000m and *MinPts* equals 8 is applied to define Cluster 1 at stop O1, and O2 as anomaly origin stops.

2) *Level 2*: Among the destination stops of each origin stop, DBSCAN algorithm differentiates the regular and anomaly destination stops. The same parameters are used to identify two clusters of destination stops at D1 and D2.

If both origin stop and destination stop are not anomaly pattern, the corresponding OD is identified as a regular spatial travel pattern. In our example, the OD pairs O1-D1 and O1-D2 are regular spatial travel patterns. Interested readers can refer to Kieu et al. (2014) for more detailed explanation of the two-level spatial travel pattern analysis.

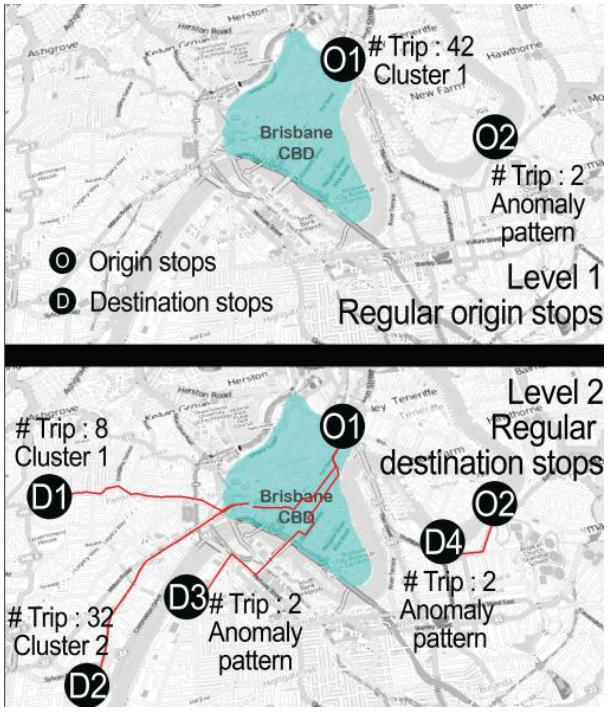


Figure 3 Two-step DBSCAN procedure for Regular OD mining

Weighted-Stop Density-Based Scanning Algorithm with Noise (WS-DBSCAN)

DBSCAN provides a good solution to identify spatial travel pattern from passenger historical itineraries. However, the major disadvantage of the DBSCAN algorithm is its quadratic computing complexity, which restricts transit operators to update individual travel pattern daily. DBSCAN takes approximately 25 hours for a Core i5, 8GB Ram personal computer to analyze & update the individual travel patterns of all passengers who has a transit journey within a day in SEQ, Australia. It means it took more than a day to update the daily changes in travel pattern, which would result in a one-day lag in transit operators understanding of their customers. Although transit authorities could employ a faster computer, running DBSCAN for daily update of individual travel pattern is an absurd task.

This section develops the theoretical foundation of the *Weighted-Stop DBSCAN* (WS-DBSCAN) algorithm to transform the quadratic complex classical DBSCAN to a problem of sub-quadratic complexity, in particular a combination of linear and quadratic complexity with fewer elements to detect and update the changes in travel pattern of individual transit passengers. The objective of WS-DBSCAN is to cluster each studied transit journey S_t to a spatial travel pattern or anomaly pattern and update the changes in travel pattern. This section only describes the process of detecting and updating regular origin patterns – the Level 1 of the two-level travel pattern analysis described in Figure 3. The Level 2 could be analyzed by WS-DBSCAN using similar method.

The following three principal features distinct WS-DBSCAN to the classical DBSCAN.

- WS-DBSCAN follows the same two-level analysis process as described in Figure 3, but instead of finding regular pattern as the first-time analysis like DBSCAN, WS-DBSCAN utilizes the existing knowledge of individual travel pattern to cluster the studied journey. Although WS-DBSCAN can also be used for first-time travel pattern analysis, it is best used for detecting and updating the changes in individual travel pattern.
- DBSCAN performs a neighborhood search from each and every point to decide if the point is a core, border or noise point. This task has a quadratic programing complexity. Conversely, WS-DBSCAN only performs neighborhood search when it is necessary (more details on the WS-DBSCAN algorithm will be demonstrated in Figure 6).
- Most of transit passengers repeatedly board or alight a transit vehicle from a stop. The classical DBSCAN treats each boarding/alighting as a unique point, which means there are overlapping points in the dataset. WS-DBSCAN significantly reduces the computation time by clustering the stops rather than the boarding/alighting itself, and gives each stop a weight i.e. the number of times the passenger boarded/alighted from that particular stop.

Following the definitions from the classical DBSCAN paper (Ester et al., 1996), we propose the basic terminologies for WS-DBSCAN.

Definition 1: The **weighted ε -neighborhood** of a transit stop p, denoted by $N_\varepsilon(p)$, is defined by

$$N_\varepsilon(p) = \sum_{q=1}^{n} W_q + W_p \mid q \in D, dist(p,q) \le \varepsilon$$

Where:

W_q is the weight of each stop q i.e. number of times the passenger boarded a transit vehicle from stop q

W_p is the weight of stop p itself,

$dist(p,q)$ is the Euclidean distance between stop p and q and

D is the stop dataset.

Definition 2: A stop p is a **core stop** if and only if $N_\varepsilon(p) \geq MinPts$

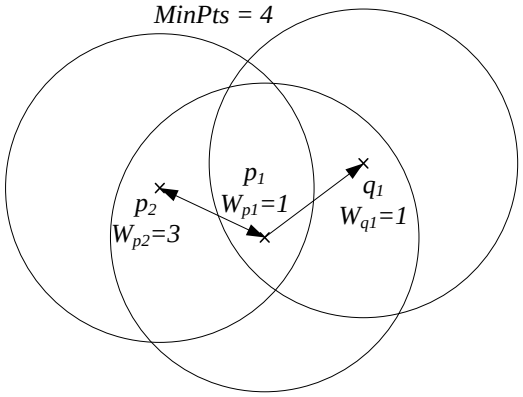
If p is a transit stop and the passenger has made more than *MinPts* number of boarding in a *weighted* ε *-neighborhood* around p, then p is called a core stop.

Definition 3: A stop q is **directly reachable** from a stop p wrt ε and *MinPts*, if

1) $q \in N_\varepsilon(p)$, and
2) $N_\varepsilon(p) \geq MinPts$ (i.e., p is a core stop)

Definition 4: A stop q is a **border stop** when it is not a *core* stop, but directly reachable from a core stop p.

Figure 4 demonstrates an example of core and border stop in WS-DBSCAN. Here for *MinPts* equals to 4, stop p_1 and p_2 are core stops, whereas q_1 is a border stop for p_1. The arrow illustrates that q_1 is *directly reachable* from p_1, but p_1 is not *directly reachable* from q_1. In contrariety, p_1 and p_2 have two-way *directly reachability* between them, because both p_1 and p_2 are core stops and lie within the ε distance of each other. A border point is *directly reachable* only from a *core* point and not from another *border* point. However, a *border* point can be *directly reachable* from multiple *core points*.



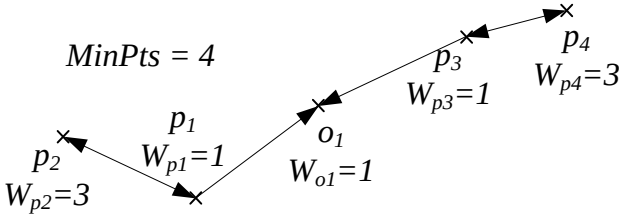
W_{pi} shows the weight of each stop
The circles show the weighted ε-neighborhood
p_1,p_2 are core stops; q_1 is a border stop
The arrows show the direct reach from a core stop

Figure 4 An example of core stop, border stop and direct reach in WS-DBSCAN

Definition 5: A stop *q* is **reachable** from a stop *p* wrt ε and *MinPts* if there is a chain of stop $p_1,\ldots,p_n$, $p_1=p$, $p_n=q$ such that p_{i+1} is directly reachable from p_i.

Definition 6: A stop *p* is **connected** to a stop *q* wrt ε and *MinPts* if there is a stop *o* such that *o* is reachable from both *p* and *q* and *vice versa*.

Figure 5 shows an example of reachable and connected stops in WS-DBSCAN. Here, p1, p2, p3 and p_4 are core points. Pont o_1 is a *border* point to both p_1 and p_3 and reachable from both p_1 and p_3. As indicated in the figure, p_1 and p_2 are *reachable*, and p_3 and p_4 are *reachable*. Therefore, p_2 and p_4 are connected through o_1.



p_1,p_2,p_3,p_4 are core stops; o_1 is a border stop
o_1 is reachable from p_2 and p_4, but p_2 and p_4 are not reachable from o_1
p_2 and p_4 are connected by o_1

Figure 5 An example of reachable and connected stops in WS-DBSCAN

Definition 7: A **cluster** *C* wrt ε and *MinPts* is a set of stops acting as a regular pattern. A cluster would have at least one *core* stop, and satisfy the following conditions:

1) $\forall p,q$: if $p \in C$ and *q* is directly reachable from *p* wrt ε and *MinPts*, then $q \in C$ (Maximality)
   This condition is to guarantee that all stops within a reachable area are considered in the same origin pattern. It also denotes that a regular pattern would have at least *MinPts* number of journeys, because cluster *C* would have at least a *core* stop *p* and its corresponding border stops *q*.
2) $\forall p,q \in C$: *p* is *connected* to *q* wrt ε and *MinPts* (Connectivity & Uniqueness)
   All stops in a cluster are at least *connected*, if not *directly reachable* or *reachable*. This condition is also to guarantee that two clusters would be distinct and stops from cluster *C* would not be connected from another cluster because if any stop is connected from two clusters, then the two clusters would be clustered as one. Thus any *core* or *border* stop would belong to one and only one cluster.

Definition 8: **Anomaly** stops wrt ε and *MinPts* are the stops that do not belong to any travel pattern. They are not *directly reachable* to any stop. Thus they are not core or border stops, and *vice versa*.

The WS-DBSCAN algorithm

This section describes the WS-DBSCAN implementation to detect a pattern for an origin stop S_t from a newly made journey and update the existing knowledge of passenger *P* travel pattern. The detection and pattern update of the destination stop from the same journey is done using similar method.

The existing travel pattern knowledge has the form of a historical stop database [S_H] for passenger *P* that stores historical origin stops with their corresponding *Weight* and *ClusterID*. Here, the *Weight* W_i of each stop S_i is the number of times the passenger has boarded a transit vehicle from that specific stop. *Cluster number* C_i of each stop *i* indicates if stop S_i is a regular ($C_i > 0, C_i \in \mathbb{Z}^+$) or anomaly stops (C_i = -1). Different positive C_i means different regular travel pattern. Table 3 shows an example of historical stop dataset [S_H] and studied stop S_t.

Table 3 Examples of: a) Historical stop dataset [S_H] and b) Studied stop S_t

a)

Historical origin stops [S_H]		
Stop S_i	*Weight* W_i	*ClusterID* C_i
O1	8	1
O2	2	1
O3	14	2
O4	1	-1
O5	2	-1

b)

Studied new stop [S_t]		
Stop S_i	Weight W_i	ClusterID C_i
S_t	1	Unknown

WS-DBSCAN detects a travel pattern by assigning a *ClusterID* for S_t

- WS-DBSCAN assigns an existing positive *ClusterID* if S_t belongs to an existing pattern
- WS-DBSCAN assigns a new positive *ClusterID* if S_t together with some other stops in [S_H] form a new pattern
- WS-DBSCAN assigns *ClusterID* equals -1 if S_t is an anomaly pattern

After each implementation, WS-DBSCAN increments the corresponding *Weights* by one and updates *ClusterID* according to the assignment of S_t's *ClusterID*. Figure 6 and the following steps describe the algorithm.

1) STEP 1: The first step is to check the sum of weights of all stops in [S_H]

$$W_{[S_H]} = \sum_{i=1}^{m} W_i \mid i \in [S_H]$$

If $W_{[S_H]} \geq MinPts - 1$ then we proceed to STEP 2. Else S_t is stored into $[S_H]$ as an anomaly stop because there is then no possibly for passenger P to have any travel pattern formed. Here, *MinPts-1* is used because the current journey contributes to a weight for the current stop.

2) STEP 2: If S_t already belongs to $[S_H]$ and has a positive *ClusterID* C_t. Its *Weight* W_t in $[S_H]$ is then incremented by one. Else the algorithm proceeds to STEP 3.
3) STEP 3: The following calculations are performed to checks if S_t could form a cluster with any stops in $[S_H]$

- Calculate the *weighted ε-neighborhood* of S_t when the dataset is $[S_H]$

  $$N_\varepsilon(S_t) = \sum_{q=1}^{n} W_q + 1 \mid q \in [S_{RN}], dist(q, S_t) \leq \varepsilon$$

  If $N_\varepsilon(S_t) \geq MinPts$, S_t is a *core* stop. There are three possibilities: a) belongs to one existing cluster; b) belongs to multiple existing clusters; and c) does not belongs to existing cluster so should be assigned a new cluster number.

  a) $\exists q \in N_\varepsilon(S_t), \mathrm{C}_q \in \mathbb{Z}^+$

  If within $N_\varepsilon(S_t)$ there is only one regular stop q, then S_t belongs to the existing cluster of q and is assigned the corresponding cluster number. All stops within its *weighted ε-neighborhood* are also assigned the same cluster number.

  b) $\exists q_1, q_2..q_n \in N_\varepsilon(S_t); \mathrm{C}_1, \mathrm{C}_2..\mathrm{C}_n \in \mathbb{Z}^+$
  $\exists \mathrm{C}_{qj} \neq \mathrm{C}_{qk} (j, k \in \{1, 2..n\})$

  If within $N_\varepsilon(S_t)$ there are multiple regular stops $q_1, q_2..q_n$ (n>1), where at least there are regular stops j and k belongs to two different clusters $\mathrm{C}_{qj} \neq \mathrm{C}_{qk} (j, k \in \{1, 2..n\})$, we merge their clusters and S_t to a combined cluster C_q because by adding S_t all these stops will be *connected* (See Definition 7). All these corresponding stops are then assigned a new combined cluster number C_q.

  c) $\nexists q \in N_\varepsilon(S_t), \mathrm{C}_q \in \mathbb{Z}^+$

  If within $N_\varepsilon(S_t)$ there is no regular stop, S_t forms a new cluster with the stops in its *weighted ε-neighborhood*. All these corresponding stops are then assigned a new cluster number.

  The corresponding *Weight* of S_t is incremented by one in each of these possibilities.

- If $N_\varepsilon(S_t) < MinPts$, S_t is not a *core* stop but it could be a *border* stop if any stop q within its *weighted ε-neighborhood* is a *core* stop.

$$N_\varepsilon(q) = \sum_{r=1}^{k} W_r + W_q \mid r \in [S_t, S_H]$$

$$\text{Where}: q = 1...n, dist(S_t, q) \le \varepsilon$$

If any $N_\varepsilon(q) \ge MinPts$, S_t is then a border stop. Similar to the previous case, S_t and its corresponding stops are assigned with an existing or new cluster number.

- If $\forall N_\varepsilon(q) < MinPts$, we can conclude that S_t is anomaly stop and store it in [S_H] with *ClusterID* equals -1. The corresponding *Weight* of S_t is incremented by one.

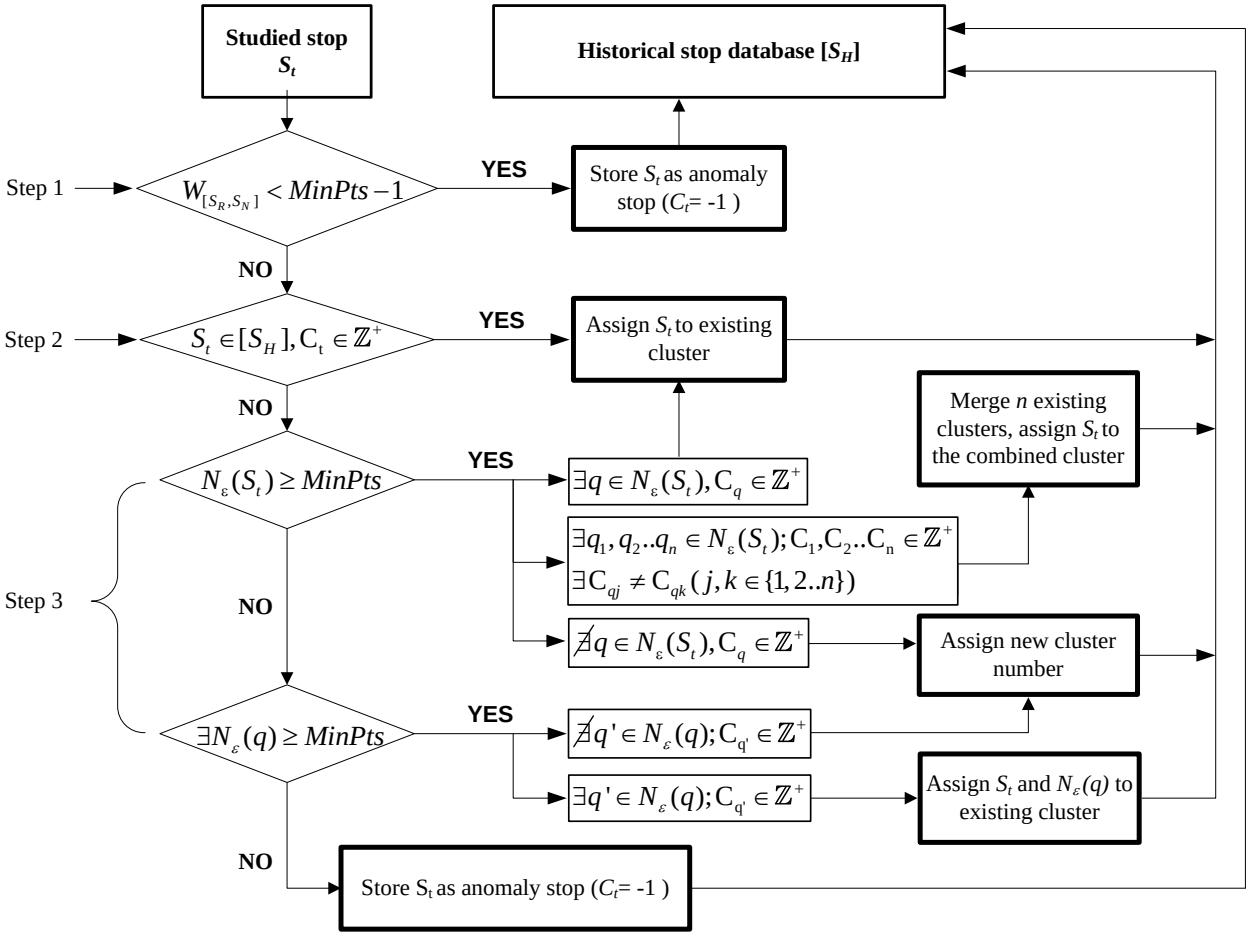


Figure 6 WS-DBSCAN algorithm

WS-DBSCAN implementation on full OD pattern detection and update

This section describes the WS-DBSCAN implementation on detecting and updating a full travel itinerary using an example. Figure 7 shows WS-DBSCAN implementations on the journeys the passenger *P* made within the day *d*.

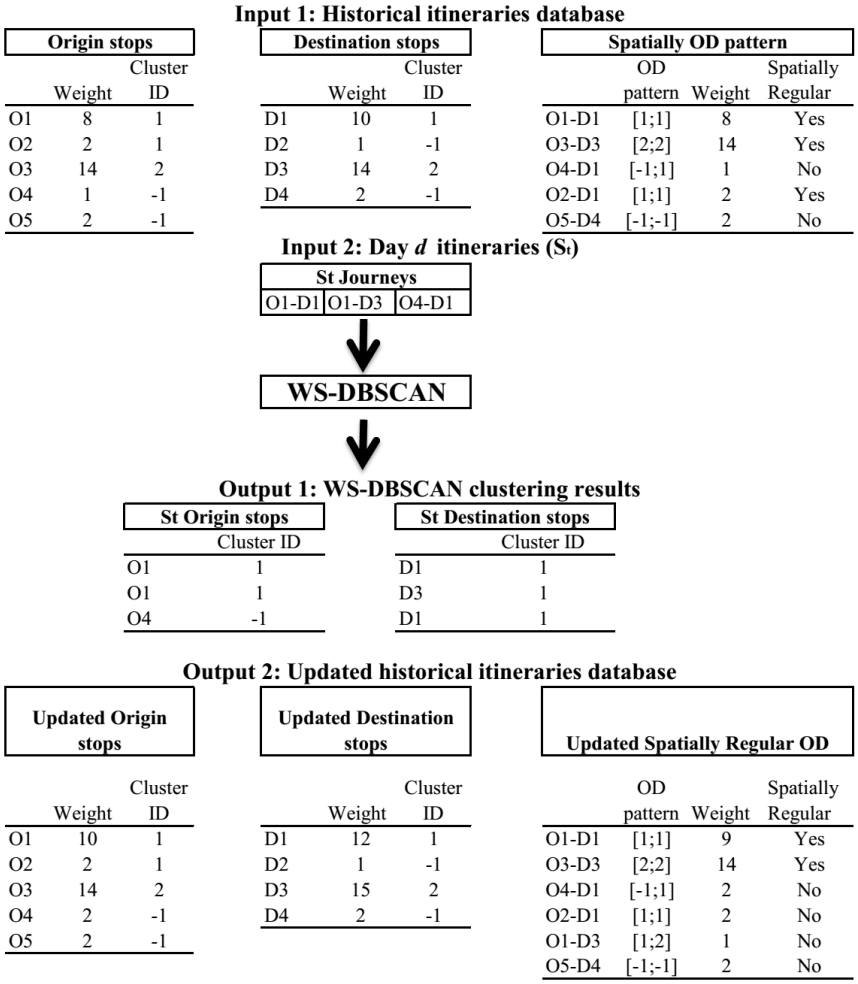


Figure 7 An example of WS-DBSCAN implementation

WS-DBSCAN requires two types of input:

- Input 1 is the historical itineraries database that stores the historical origin stops, destination stops; and a spatially regular OD database that stores the full OD itineraries. Each historical origin and destination stop is associated with a *Weight* and a *ClusterID*. Cluster number differentiates the regular origin/destination patterns and anomaly pattern. Each OD itinerary in the Spatially OD pattern database is also associated with a *Weight*, a variable *OD pattern* and a binary variable indicating regular/anomaly OD pattern. The *OD pattern* stores the *Cluster numbers* of both origin and destination stop in the format [$O_{Clusternumber}$;$D_{Clusternumber}$].
  Here O1 and O2 both belong to origin cluster 1, so both O1-D1 and O2-D1 have [1;1] for their *OD pattern*. We define an OD pattern *i* as a regular OD pattern if the total *Weight* of its *OD Pattern* is larger than *MinPts.*
- Input 2 is the journeys that passenger *P* made during the studied day, where *P* made three journeys within the day *d* in this particular example. Each OD itinerary in Input

2 is fed into the WS-DBSCAN algorithm. Here, passenger P made 3 journeys O1-D1, O1-D3 and O4-D1 during the study day *d*.

WS-DBSCAN provides two sets of output. While Output 1 represents the pattern detection, Output 2 represents the pattern update feature of WS-DBSCAN.

- Output 1 is the assigned cluster number of each itinerary in Input 2. Here O1, D1 and D3 are identified as regular stops and O3 as anomaly stop.
- Output 2 takes Output 1 results and shows the updated spatial travel pattern in terms of origin/destination and OD pattern. Here among the three journeys passenger *P* made within the day *d*, O1-D1 is clearly a spatial regular OD pattern and O4-D1 and O5-D4 are clearly an anomaly patterns. Although both O1 and D3 are regular stops, the pair O1-D3 is an anomaly pattern because its *OD Pattern* does not have a *Weight* higher than *MinPts*.

Numerical experiment

This section validates the applicability of WS-DBSCAN to the real data, in comparison with the classical DBSCAN algorithm.

Experiment setup

The numerical experiment is set to emulate a working environment to compare the performance of classical DBSCAN and WS-DBSCAN. The last working week (Monday to Friday) of June, 2012 has been used as the testing dataset, whereas the other weeks acted as the historical itineraries data. The experiment emulates a working environment by the following three steps

1) At the end of each day in the testing dataset (last working week of June 2012), we collect all Smart Card transactions of the day
2) The algorithm illustrated in Figure 1 reconstructs individual transactions into passenger itineraries
3) The classical DBSCAN and WS-DBSCAN analyzes the studied day itineraries and relates with the historical itineraries data to detect and update the travel pattern. The implementation of DBSCAN and WS-DBSCAN has been described in Figure 3 and **Figure 7**, respectively.

We adopt the same example of detecting and updating passenger *P* travel pattern after day *d* in **Figure 7** to compare the two algorithms. While Input 1 and Input 2 in Figure 8 are fed into WS-DBSCAN, historical OD journeys similar to Table 2 and Input 2 are fed into DBSCAN.

Experiment results

Table 4 shows the comparison between the time efficiency of DBSCAN and WS-DBSCAN. We compare the two performances by calculating the ratio between WS-DBSCAN computation time and DBSCAN computation time.

$$Q_t = \frac{t_{WSDBSCAN}}{t_{DBSCAN}} \times 100\%$$

Where t_{DBSCAN} and $t_{WSDBSCAN}$ are the computation time of the DBSCAN and WS-DBSCAN implementation, respectively.

Table 4 Computation time comparison of DBSCAN and WS-DBSCAN to detect and update each studied journey.

Algorithms	Mean(s)	Median(s)	Mean Regular Detection(s)	Mean Anomaly Detection(s)
DBSCAN	23.34×10^{-2}	23.23×10^{-2}	23.52×10^{-2}	22.72×10^{-2}
WS-DBSCAN	10.78×10^{-4}	10.47×10^{-4}	10.23×10^{-4}	11.17×10^{-4}
Q_t	0.46%	0.45%	0.43%	0.49%

Mean and Median of computation time are for a two-level analysis detection & update for a studied journey in the testing dataset in seconds. The time is counted from an itinerary is presented, until the update process is completed.

WS-DBSCAN costs only around 0.46% in computation time compared to the classical DBSCAN, while provides the same mined travel pattern results as DBSCAN by sharing the same clustering method and parameters. Table 5 shows the observed computation time for detecting and updating the travel pattern of all passengers travelled on each day from 18th to 22nd June 2012.

Table 5 Travel pattern detecting and updating time for each testing day

Day	18th Jun	19th Jun	20th Jun	21st Jun	22nd Jun
DBSCAN	23.59	24.79	25.29	25.72	25.30
WS-DBSCAN	0.11	0.11	0.12	0.12	0.12
Q_t	0.46%	0.44%	0.47%	0.47%	0.47%

Computation time is counted in hours. The analysis has been performed on all travelled passengers on each day

While DBSCAN took approximately a day to detect and update the changes in spatial travel pattern of all passengers travelled on each testing day, WS-DBSCAN takes 6-7 minutes of computation time. Figure 8 illustrates the computation time of WS-DBSCAN when the passenger has different number of journeys and number of OD pairs in the historical dataset. The computation time varies from 8.5×10^{-4} to 12.5×10^{-4} seconds. The WS-DBSCAN is exceptionally fast when the number of journeys in the historical dataset is large and the number of OD pairs is small. The algorithm provides a decision in the shortest time when the number of journeys is less than *MinPts*, because the studied journey is definitely an anomaly pattern. The algorithm is only slightly slower for passengers whose the number of journeys is close to the number of OD pairs. These passengers only account for less than 3% of the population.

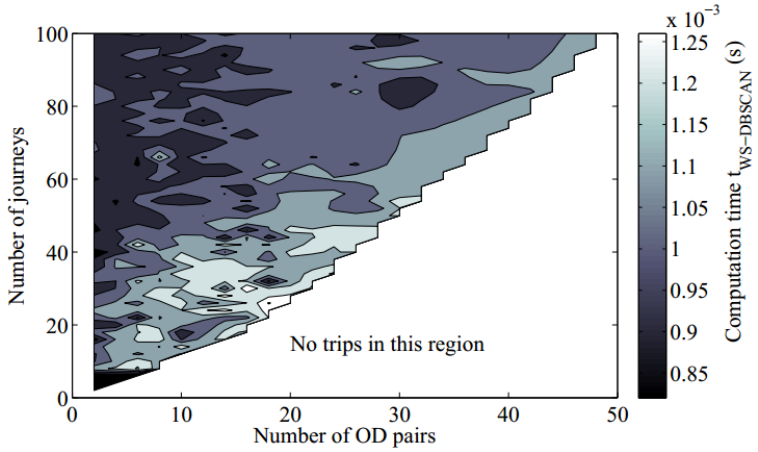
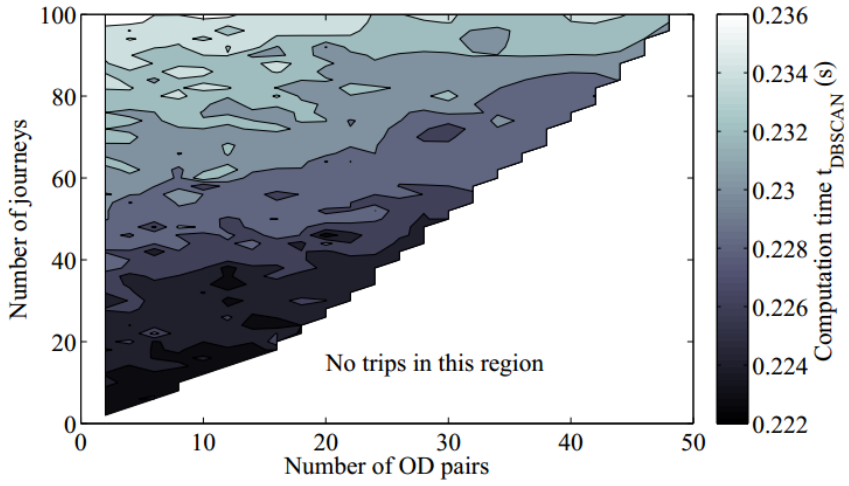
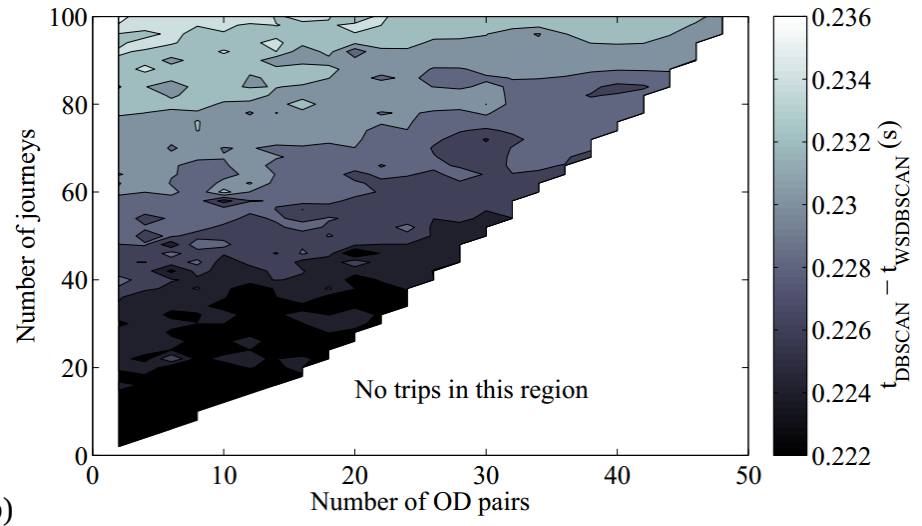


Figure 8 WS-DBSCAN computation time

WS-DBSCAN time-effectiveness is further illustrated in comparison with the DBSCAN computation time. Figure 9(a) shows that DBSCAN cost more as the number of journeys increases, reflecting its quadratic computation complexity. The difference between the computation time of DBSCAN and WS-DBSCAN algorithm on the same problem is demonstrated in Figure 9(b). The figure shows that the time savings by using WS-DBSCAN instead of DBSCAN is greater as the number of journeys is getting larger.

(a)





(b)

Figure 9 DBSCAN computation time: a) in seconds, b) in comparison with WS-DBSCAN

The numerical experiment demonstrates that WS-DBSCAN is more than 200 times faster while provides the same travel pattern analysis as DBSCAN.

Sensitivity analysis of WS-DBSCAN parameters

WS-DBSCAN requires the same set of parameters same as in DBSCAN: the minimum number of journeys *MinPts* and the density reach distance ε. These parameters decide the detection result in WS-DBSCAN and DBSCAN which have to be chosen carefully before implementation.

The maximum density reach distance ε denotes the walking distance of the passenger from one to another stop of the same journey pattern. For instance a student might randomly board from stop A or stop B to travel to school, and ε is the distance between these two stops. The Transit Capacity and Quality of Service Manual shows that 90% of passengers would walk less than 500m to transit stops (TRB, 2013). If we increase ε, more stops will be considered as regular origin stop. ε should not be too large since both origin and destination might be clustered into the same boarding pattern if ε is larger than the travel distance. ε can be found by travel survey and vary from case to case. Burke and Brown (2007) found that people in Brisbane and Perth, Australia walk significantly longer than people in US cities. The definition of ε is then not simply the walkable area but for each passenger it is the "preferable walking distance" or "activity area". The value of ε could be different for passengers who prefer long and short walk.

The examination of *MinPts* can be broken down into how transit operators define travel pattern. Given the number of journeys over a study period, *MinPts* is equal to the minimum journeys made to be considered as "regular". As *MinPts* increases, less journeys are identified as regular travel patterns. A value of *MinPts* equal to 2 means that any repeated boarding will be considered as regular. Figure 10 illustrates the *MinPts* and ε sensitivity analysis results. The figure provides guidance on how to define spatial travel pattern.

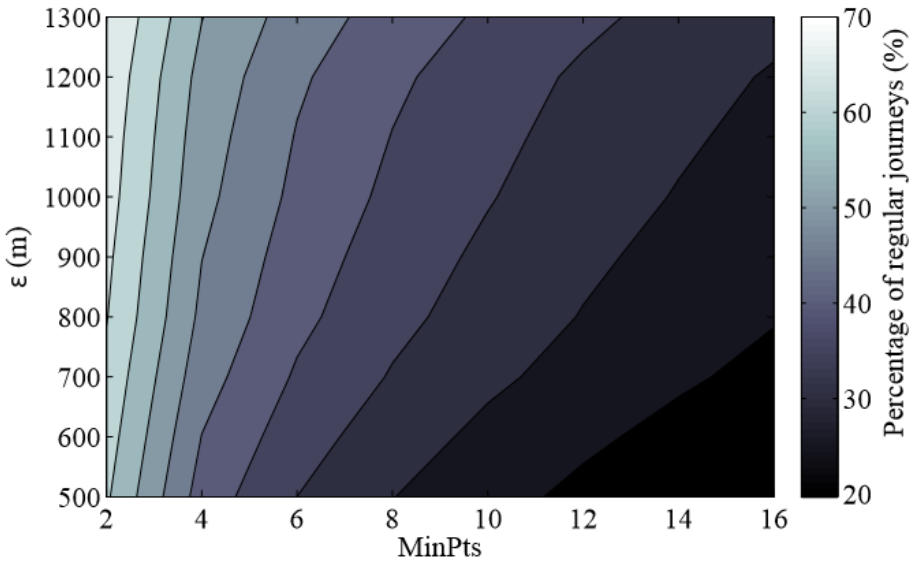


Figure 10 Sensitivity analysis of *MinPts* and ε

MinPts should also be chosen considering the observation period of historical itineraries data. As the number of the total journeys increases as the observation period becomes longer, the value *MinPts* is only valid for a specific period. Figure 11 demonstrates the percentage of regular journeys at different period when *MinPts* varied from 2 to 16, and is equal to 1000 m.

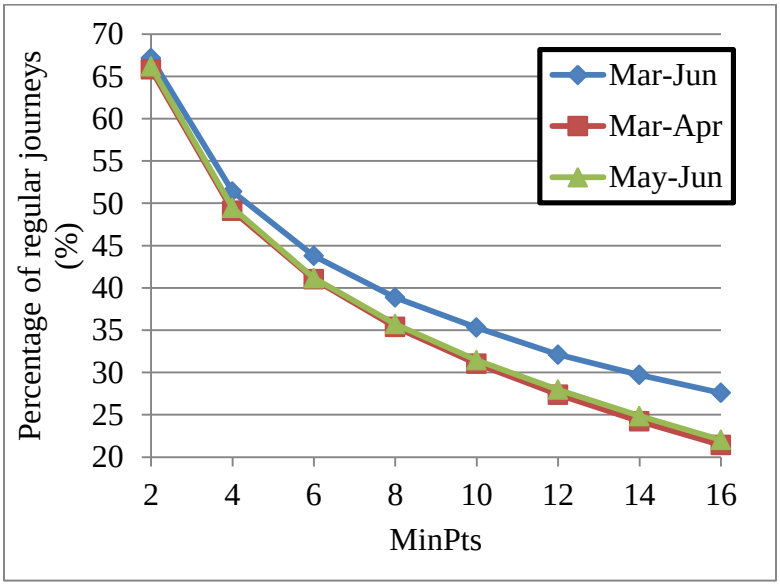


Figure 11 Sensitivity analysis of *MinPts* at different observation period

Different values of *MinPts* have been used on the whole dataset (Mar to Jun 2012) and two shorter periods (Mar to Apr 2012 and May to Jun 2012). Shorter observation periods show lower share of regular journeys, especially when *MinPts* becomes larger. It is because there are less spatial travel patterns that are repeated for more than *MinPts* times within a shorter period. In

practice the observation period should be fixed and should not be too long, because passenger travel pattern can change significantly during a long time period. The historical itineraries data can be store for a period of several months in a rolling horizon process, i.e. detecting and updating the passenger individual travel pattern data in each period. Figure 11 also shows that the regular journeys shares are similar at different *MinPts* when the observation periods are equal.

DBSCAN and WS-DBSCAN provide exactly the same travel pattern analysis results when using the same set of parameters (*MinPts* and ε). The accuracy of these algorithms when compared to the actual transit passenger travel pattern solely depends on these parameters. A short survey could solve the problem of choosing the best parameters or validate the mining results. In the meantime, the sensitivity analysis of *MinPts* and ε as discussed in this paper provides insights into the sensitivity of these parameters.

Conclusion

This paper proposes the Weighted-Stop DBSCAN (WS-DBSCAN) algorithm to detect and update spatial travel pattern in much less computation time than the classical DBSCAN algorithm. Compared to the classical DBSCAN, the proposed WS-DBSCAN algorithm has a logical mechanism suited for transit spatial travel pattern analysis. Each stop in the historical dataset is given a weight to reduce the neighborhood search, and only and only perform compulsory neighborhood search. WS-DBSCAN also embraces the existing knowledge of individual passenger to reduce travel pattern analyzing time. WS-DBSCAN on average detects and updates a new transit journey in 0.45% of what it costs for classical DBSCAN.

The methodology presented in this paper assists transit operators to observe and record the daily changes in individual travel pattern. While the classical DBSCAN takes 24-25 hours for a travel pattern analysis of all travelled passengers for a day, WS-DBSCAN takes less than 7 minutes to perform the same analysis. WS-DBSCAN enables transit agencies to use individual travel pattern on a daily basis, such as in customized service provision and operational strategy. Although larger dataset than the SEQ would definitely take more time for a complete analysis, employing multiple computers to analyze each passenger segment would easily solve this problem.

The introduction of WS-DBSCAN also enables us to investigate the possibility of using travel pattern data in real-time. Although most of the Smart Card systems are not associated with user contact information, individual travel pattern can be stored through Smart Card ID. Transit passengers can use a smartphone application to get customized real-time information from service providers for the service they regularly use. The information is given though the input of Smart Card ID and a proof-of-card ownership by the barcode behind each card. By managing the customized information by the unique Smart Card ID, the information and service can remain oriented for each individual and at the same time, maintain privacy. However, various practical

issues have to be solved before individual travel pattern can be used in real-time, such as the computation & telecommunication constraints, which information to be given and who to be enrolled in the system. These problems lay out immediately accessible avenues for future research. In the meantime, WS-DBSCAN can be used to detect and update individual travel pattern for much less cost than the classical DBSCAN.

WS-DBSCAN algorithm focuses only on spatial travel pattern analysis. Temporal and spatial travel pattern can also be simultaneously analyzed in a two-step approach, where the data is first clustered into temporal travel pattern based on the time of the day and thereafter, within the time period spatial pattern can be evaluated. An extension of WS-DBSCAN to consider the temporal attributes will reduce the analysis sample size and reveals temporal travel pattern. The joint consideration of spatial and temporal travel pattern is also a future research direction.

Acknowledgment

This research is supported by Queensland University of Technology. The AFC data used in this study was provided by Translink, the transit authority of South East Queensland, Australia. The authors would like to thank the two anonymous referees and the Editor-in-Chief of Transportation Research Part C for their critical comments. The conclusions in this paper reflect the understandings of the authors, who are responsible for the accuracy of the data.

Reference

Burke, M., Brown, A., 2007. Distances people walk for transport. Road & Transport Research: A Journal of Australian and New Zealand Research and Practice 16(3), 16.

Chu, K.K.A., Chapleau, R., 2010. Augmenting Transit Trip Characterization and Travel Behavior Comprehension: Multiday Location Stamped Smart Card Transactions. Transportation Research Record: Journal of the Transportation Research Board 2183(1), 29-40.

Chu, K.K.A., Chapleau, R., Trepanier, M., 2009. Driver-Assisted Bus Interview. Transportation Research Record: Journal of the Transportation Research Board 2105(1), 1-10.

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise, Second international conference on knowledge discovery and data mining. Amer Assn for Artificial, p. 226.

Hasan, S., Schneider, C., Ukkusuri, S., González, M., 2013. Spatiotemporal Patterns of Urban Human Mobility. J Stat Phys 151(1-2), 304-318.

Jang, W., 2010. Travel time and transfer analysis using transit smart card data. Transportation Research Record: Journal of the Transportation Research Board 2144(1), 142-149.

Kieu, L.M., Bhaskar, A., Chung, E., 2014. Passenger Segmentation Using Smart Card Data. IEEE Transactions on Intelligent Transport System (In Press).

Lee, S., Hickman, M., 2013. Are Transit Trips Symmetrical in Time and Space? Transportation Research Record: Journal of the Transportation Research Board 2382(-1), 173-180.

Lee, S., Hickman, M., 2014. Trip purpose inference using automated fare collection data. Public Transp 6(1-2), 1-20.

Lee, S.G., Hickman, M., Tong, D., 2012. Stop Aggregation Model. Transportation Research Record: Journal of the Transportation Research Board 2276(1), 38-47.

Ma, X., Wu, Y.-J., Wang, Y., Chen, F., Liu, J., 2013. Mining smart card data for transit riders' travel patterns. Transportation Research Part C: Emerging Technologies 36, 1-12.

Morency, C., Trepanier, M., Agard, B., 2007. Measuring transit use variability with smart-card data. Transport Policy 14(3), 193-203.

Pelletier, M.P., Trépanier, M., Morency, C., 2011. Smart card data use in public transit: A literature review. Transportation Research Part C: Emerging Technologies 19(4), 557-568.

Seaborn, C., Attanucci, J., Wilson, N., 2009. Analyzing Multimodal Public Transport Journeys in London with Smart Card Fare Payment Data. Transportation Research Record: Journal of the Transportation Research Board 2121(-1), 55-62.

Translink, 2007. How to use your go card on the TransLink network: TransLink go card user guide (part 1 of 2).

TRB, 2013. Transit Capacity and Quality of Service Manual 3rd Edition. Transportation Research Board.

Utsunomiya, M., Attanucci, J., Wilson, N., 2006. Potential Uses of Transit Smart Card Registration and Transaction Data to Improve Transit Planning. Transportation Research Record: Journal of the Transportation Research Board 1971(-1), 119-126.