Filling Delivery Time Automatically Based on Couriers' Trajectories

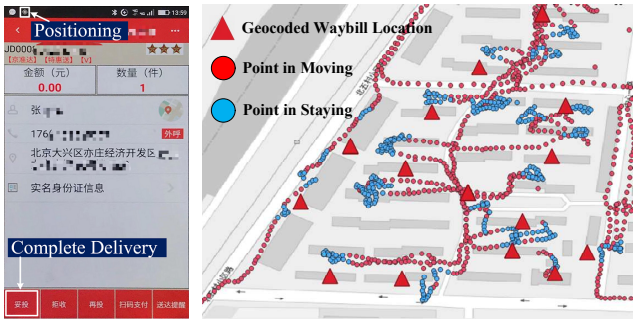
Sijie Ruan, Xi Fu, Cheng Long, *Member, IEEE,* Zi Xiong, Jie Bao, Ruiyuan Li, Yiheng Chen, Shengnan Wu, and Yu Zheng, *Fellow, IEEE*

Abstract—Nowadays, couriers are still the main solution to address the "last mile" problem in logistics. They are usually required to record the delivery time of each parcel manually, which is essential for delivery insurances, delivery performance evaluations, and customer available time discovery. Stay points extracted from couriers' trajectories provide a chance to fill the delivery time automatically to ease their burdens. However, it is challenging due to inaccurate delivery locations and various stay scenarios. To this end, we propose the improved Delivery Time Inference (DTInf+), to infer the delivery time of waybills based on their trajectories. DTInf+ is composed of three steps: 1) *Data Pre-processing*, which organizes waybills and stay points by delivery trips, 2) *Delivery Location Mining*, which obtains the delivery location for each address and each Geocoded waybill location by mining historical delivery caused stay points, and 3) *Delivery Event-based Matching*, which infers the delivery caused stay point for waybills at the same delivery location based on a pointer network-like model SPSelector to obtain the delivery time. Extensive experiments and case studies based on real-world datasets from JD Logistics confirm the effectiveness of our approach. Finally, a system is deployed in JD Logistics.

Index Terms—Trajectory Data Mining, Trajectory Annotation, Urban Computing

1 Introduction

Express couriers are the main solution to address the "last mile" problem in logistics, currently. With the active development of e-commerce, the workloads of couriers become heavier. When delivering a parcel, the courier is asked to perform an important additional task besides the pick-ups and deliveries, i.e., recording the delivery time of each parcel. Figure 1(a) shows the interface of a courier's PDA, displaying detailed information and actions for a parcel delivery task, called *waybill*. The "Complete Delivery" button is required to be clicked immediately when the parcel is delivered. While the task of clicking this button each time a parcel is delivered looks tedious, it records the delivery time that is vital for many applications in JD.com, e.g., delivery insurances, delivery performance evaluations, and customer available time discovery.



(a) Courier's PDA. (b) Courier's Trajectories & Waybills.

Fig. 1. Background and Opportunities.

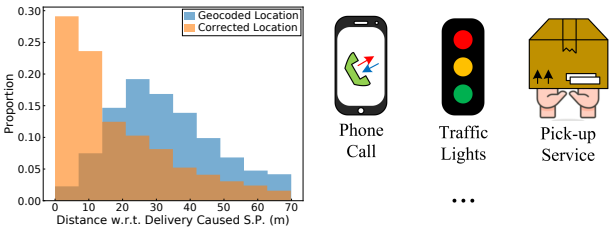
Therefore, it would be of great value for both the couriers and the logistics company if we can fill the delivery time automatically. Fortunately, a courier's PDA also records his/her locations during the working hours as shown in Figure 1(a), which provides a chance to infer the delivery time. Intuitively, a courier would stay at a location for a while when he/she is delivering a parcel, thus generating a stay point [2]. A straightforward solution would be to extract the delivery time based on stay points of trajectories.

However, in our preliminary data analysis, there are many exceptions between the stay points and delivery locations. For example, Figure 1(b) shows the point distribution of a trajectory in a region, where circles are GPS points of a courier's trajectory; and triangle markers are the Geocoded waybill locations, which are parsed from the plain text shipping addresses via Geocoding services[1]. According to the figure, there are many more trajectory stay points than the

- *This paper is an extended version of an earlier paper published at the 26th SIGKDD Conference (KDD 2020) [1].*
- *S. Ruan and Y. Zheng are with the School of Computer Science and Technology, Xidian University, Shaanxi, China. S. Ruan and Y. Zheng are also with JD Intelligent Cities Research, China and JD Intelligent Cities Business Unit, JD Technology, Beijing, China. E-mail: sjruan@stu.xidian.edu.cn, msyuzheng@outlook.com*
- *X. Fu is with Shanghai Jiao Tong University, Shanghai, China. Work was done when X. Fu was an intern at Nanyang Technological University. E-mail: fuxi1128@sjtu.edu.cn*
- *C. Long is with Nanyang Technological University, Singapore. E-mail: c.long@ntu.edu.sg*
- *Z. Xiong is with Wuhan University, Hubei, China. E-mail: zixiong@whu.edu.cn*
- *J. Bao is with JD Intelligent Cities Research, China and JD Intelligent Cities Business Unit, JD Technology, Beijing, China. E-mail: baojie@jd.com*
- *R. Li is with the College of Computer Science, Chongqing University, Chongqing, China. R. Li is also with JD Intelligent Cities Research, China and JD Intelligent Cities Business Unit, JD Technology, Beijing, China. E-mail: liruiyuan@whu.edu.cn*
- *Y. Chen and S. Wu are with JD Logistics, Beijing, China. E-mail: {chenyiheng,wushengnan1}@jd.com*
- *Y. Zheng and J. Bao are corresponding authors.*

Manuscript received October, 2020; revised June, 2021; accepted July, 2021.

1. https://en.wikipedia.org/wiki/Geocoding



(a) Inaccurate Delivery Locations. (b) Various Stay Scenarios.

Fig. 2. Challenges to Infer the Delivery Time.

Geocoded waybill locations, and many Geocoded waybill locations are not always close to the stay points. Thus, it is not possible to infer the delivery time directly from stay points due to the following two main challenges:

- **Inaccurate delivery locations.** According to the delivery time of each waybill annotated by couriers, we can find its delivery caused stay point in the trajectory. We plot the distribution of the distance between the Geocoded waybill location and such stay point in Figure 2(a). It shows that most of the Geocoded locations have some distance shifts to the delivery caused stay points[2]. Therefore, for each waybill, we can not treat the Geocoded location as the delivery location, and infer the delivery time based on the closest stay point to it.
- **Various stay scenarios.** Even if we find the closest stay point to the actual delivery location, we still cannot say that the parcel is delivered at that stay point. The reason is that a courier might stay at a location for various reasons. As shown in Figure 2(b), a courier might stay when he is calling the customers, waiting for the traffic lights, or picking up parcels from customers.

In the early version [1] of this work, we tackle the automatic delivery time filling by designing a Delivery Time Inference (DTInf) system, which solves the above challenges by mining the delivery location for each waybill and modeling the likelihood of delivery caused stay points to distinguish them from other stay points that are generated due to other reasons. In the offline phase, for each Geocoded waybill location l_a, we find historical delivery caused stay points, which correspond to deliveries to addresses whose Geocoded location is l_a, and use the spatial centroid of those stay points as the delivery location of l_a. In the online inference phase, we first retrieve the delivery location for each waybill based on its Geocoded location, then partition waybills with the same delivery location into several groups, each corresponding to a delivery event. After that, for each event, we use an MLP [3] to compute a score for each neighboring stay point, indicating the likelihood that the stay point is the delivery caused stay point of the event. Finally, the stay point with the highest score is selected as the inferred delivery caused stay point, and its time is used as the inferred delivery time for waybills of the event.

Nevertheless, there is still room of improving DTInf in the following two aspects. 1) When mining delivery location, DTInf assumes that each Geocoded waybill location corresponds to one delivery location. However, we further find there could be multiple delivery locations for a Geocoded waybill location, e.g., addresses in different buildings might be parsed to the same Geocoded location given incomplete POI database of Geocoding. Under such assumption, addresses whose Geocoded locations are the same but actual delivery locations are different would be corrected to a single inferred delivery location, which is inaccurate, and affects the delivery caused stay point modeling. 2) When performing the inference, for each delivery event, we independently classify whether each nearby stay point is the delivery caused stay point of the event, ignoring the existence of other nearby stay points. This fails to capture some correlation among stay points. For example, a courier might visit a location for multiple times in a trip, and each visit would generate a stay point. According to the domain knowledge, parcels are more likely to be delivered when he visits that location for the first time.

Maintaining the design principles of DTInf, in this paper, we present DTInf$^+$ which advances the DTInf framework with the following two improvements. 1) In addition to inferring the delivery location for each Geocoded waybill location, we also infer it for each address. During the online inference, the delivery location knowledge from addresses would be given a higher priority to be used than that from Geocoded locations if addresses have ever appeared, since each address usually corresponds to one actual delivery location. 2) Instead of independently classifying each stay point, we propose a pointer network [4] based model SPSelector, which jointly considers all candidate stay points of a delivery event to make the prediction effectively.

To summarize, we make the following contributions:

- We present a novel automatic delivery time filling problem based on trajectories and identify its challenges.
- We propose a delivery time inference solution DTInf$^+$, which not only overcomes the distance shifts of delivery locations, but also considers various factors. A new delivery location correction strategy and a stay point joint selection model are proposed to improve DTInf.
- Experiments as well as case studies on real-world datasets from JD Logistics show the effectiveness of DTInf$^+$, which outperforms the best baseline by 52.8%.
- A system based on DTInf$^+$ is deployed in JD Logistics and used internally.

2 Overview

2.1 Preliminaries

Definition 1 (Waybill). A waybill is a parcel delivery task assigned to a courier, denoted as a 5-tuple $w = (addr, l_a, \mathbf{F}_p, t_{re}, t_d)$. $addr$ is the shipping address, l_a is the Geocoded location of $addr$, $\mathbf{F}_p$ are features of the parcel, e.g., the weight and the volume, t_{re} is the timestamp, at which a courier receives the parcel, and t_d is the delivery time.

We note that the shipping address in plain text is not available for us due to privacy protection issues, so we use the combination of the customer ID and the Geocoded waybill location of an address to act as the identifier of the address. The delivery time t_d normally needs to be manually recorded by couriers. In this study, we aim to automatically fill t_d so as to reduce couriers' burden of recording it.

2. In our study region, there does not exist express lockers. If parcels are delivered to lockers, the shifts could be even larger.

Definition 2 (Delivery Location). A delivery location is a spatial point, denoted as $l_d = (x, y)$ (i.e., longitude and latitude), where a courier gives the parcel to the corresponding customer, or leaves it at an express cabinet.

Definition 3 (Trajectory). A trajectory is a sequence of spatio-temporal points, denoted as $T =< p_1, p_2, ..., p_n >$, where each point $p = (x, y, t)$ indicates the physical presence at a location (x, y) at time t. Points in a trajectory are organized chronologically.

Definition 4 (Stay Point). A stay point is a subsequence of the trajectory, which semantically means that a moving object stays in a geographic region for a while. Formally, given a distance threshold D_{max} and a time threshold T_{min}, $< p_i, p_{i+1}, ..., p_j >$ is called a stay point sp if $distance(p_i, p_k) \leq D_{max} (\forall k \in [i+1, j])$, $distance(p_i, p_{j+1}) > D_{max}$ (if $j < n$), and $|p_j.t - p_i.t| \geq T_{min}$. The time interval of a sp is $[p_i.t, p_j.t]$.

The location of a sp is estimated using its spatial centroid:

$$sp.x = \frac{\sum_{k=i}^{j} p_k.x}{j-i+1} \quad \text{and} \quad sp.y = \frac{\sum_{k=i}^{j} p_k.y}{j-i+1} \tag{1}$$

The time of a sp is defined as the middle point of its time interval:

$$sp.t = p_i.t + \frac{p_j.t - p_i.t}{2} \tag{2}$$

Particularly, if a stay point is caused by a delivery, we call it a *delivery caused stay point*. In historical data, it can be identified by checking whether there is a parcel delivered during the time interval of the stay point based on the delivery time of the waybill.

Definition 5 (Delivery Trip). A delivery trip is a process that a courier delivers a batch of parcels to customers.

2.2 Problem Definition

We aim to infer the delivery time for parcels in each delivery trip based on stay points in couriers' trajectories. We assume that a courier should report the parcels failed to be delivered (or successfully delivered) after the trip. Therefore, we know those waybills whose delivery time could be inferred.

We propose to identify for a waybill the stay point, at which the parcel of the waybill is delivered, and then use its time as the inferred time of the waybill. There are two reasons for this strategy:

- *Short delivery stay*: According to delivery caused stay points, the average delivery duration is 13 minutes, and for 80% waybills, the delivery duration does not last for longer than 20 minutes as shown in Figure 3(a). Such granularity is acceptable for target applications, e.g., customer available time discovery.
- *Anonymized shipping address*: One or more parcels can be delivered at the same stay point. However, Geocoding anonymizes the detailed floor information of addresses as shown in Figure 3(b). Therefore, there is no guidance for deciding the orders by which the waybills are finished. Thus, it is impossible to infer finer-grained time.

Therefore, the problem of filling the delivery time of a waybill is transformed to be one of identifying its delivery caused stay point. We define the problem as follows:

Given courier's stay points $SP = \{sp_j | j \in 1, .., m\}$ detected from the trajectory of a delivery trip, and the waybills $W = \{w_i | i \in 1, ..., p\}$ he/she completed in the trip, we aim to match each waybill w_i with its delivery caused stay point sp_j.

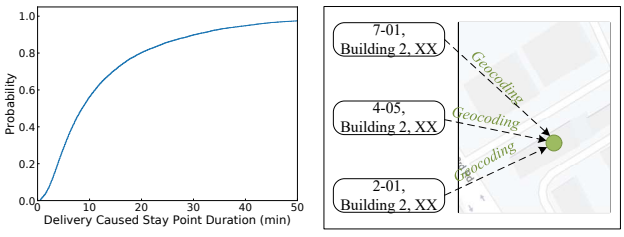
2.3 System Framework

The system framework of DTInf+ is elaborated in Figure 4, consisting of three components:

Data Pre-processing. This component takes couriers' trajectories and waybills and performs three main tasks: 1) *Noise Filtering*, which removes the outlier GPS points; 2) *Stay Point Detection*, which detects all the stay points from the trajectories; 3) *Delivery Trip Identification*, which separates waybills and stay points by the identified delivery trips (detailed in Section 3).

Delivery Location Mining. This component takes historical waybills and stay points, and generates the location mapping from the shipping address to the delivery location. In case that a new shipping address might appear in the future, we also generate a delivery location mapping from the Geocoded waybill location so that delivery location knowledge can be generalized to unseen shipping addresses. It includes three steps: 1) *Inverted Indexing*, which finds all historical delivery caused stay points for each shipping address or Geocoded waybill location; 2) *Location Inference*, which infers the raw delivery locations based on historical delivery caused stay points; 3) *Location Refinement*, which reduces the redundant raw delivery locations by clustering neighboring raw delivery locations (detailed in Section 4).

Delivery Event-based Matching. This component takes the stay points and waybills in a trip, and identifies the most likely delivery caused stay point for each waybill. Two steps are conducted: 1) *Delivery Event Construction*, which uses the



(a) Delivery Stay Duration. (b) Anonymized Addresses.

Fig. 3. The Motivations of Problem Formulation.

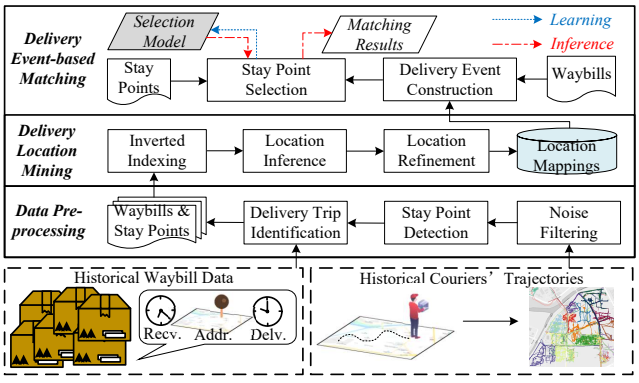


Fig. 4. System Framework.

location mapping to group waybills based on the corrected delivery location, where each group of waybills is called a delivery event; 2) *Stay Point Selection*, which infers the delivery caused stay point for each delivery event, and the time of the selected stay point is used as the inferred delivery time for waybills in that event (detailed in Section 5).

3 Data Pre-processing

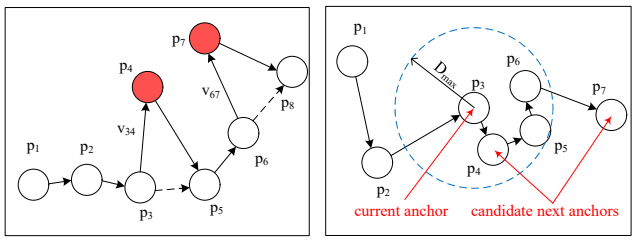
In this component, trajectories are cleaned and stay points are extracted. Then the stay points and waybills are separated and organized by identified delivery trips.

3.1 Noise Filtering

The trajectories generated by a courier's PDA usually contain noise points. For example, as shown in Figure 5(a), the error of p_4 and p_7 might be several hundred meters away from its actual location. Such noise points would affect the quality of stay point detection. A heuristic-based approach proposed in [5] is used to filter noise points in trajectories. The algorithm sequentially calculates the traveling speed for each point in a trajectory based on its precursor and itself. If the speed is larger than a threshold, the current examined point is removed from the trajectory. In this example, if v_{34} and v_{67} are larger than the speed threshold, they are removed from the trajectory. The speed threshold is set to 54km/h since the moving speed of a courier would rarely exceed this threshold.

3.2 Stay Point Detection

Based on the cleaned trajectories, we extract all stay points from them. We use stay points not only to infer the delivery time, but also to find the actual delivery locations. The stay point detection algorithm proposed in [2] is employed. The algorithm first checks if the distance between an anchor point and its successors in a trajectory is larger than a given threshold D_{max}. In the example shown in Figure 5(b), p_3 is the current anchor point, and p_4 to p_6 are its successors within D_{max}. It then calculates the duration between the anchor point and the last successor within D_{max} (p_3 and p_6). If the duration is larger than the given temporal threshold T_{min}, a stay point is detected (p_3 to p_6), and the anchor point moves to the next point after the current stay point (p_7). Otherwise, the anchor point moves forward by one (p_4). This process is repeated until the anchor point moves to the end of the sequence. The algorithm has the chance to generate stay points that are temporally consecutive, which makes little difference for the delivery time inference. Therefore, we also merge those consecutive stay points. We tried different parameter combinations and found that most delivery time of waybills can be included in stay points when we set $D_{max} = 20$m and $T_{min} = 30$s.



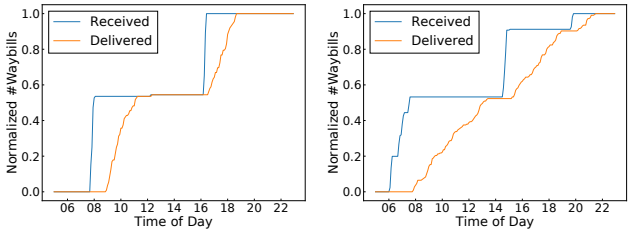
(a) Noise Filtering. (b) Stay Point Detection.

Fig. 5. Trajectory Pre-processing.

3.3 Delivery Trip Identification

According to workloads, a courier can have one, two, or several delivery trips each day. For example, a normal workday usually contains 2 trips, while a promotion day (e.g., "6.18", Double 11) might contain 3∼4 trips given the tremendous workloads. A courier will start a delivery trip after he/she receives the newly arrived parcels. Figure 6 shows the (normalized) number of parcels received and delivered by a courier during a day for a normal workday and a promotion day. It is noticeable that the number of sharp increases and the time of sharp increases of the parcel receiving curve are dynamic due to the upstream logistics arrangements, which further influences the start time of the delivery trip.

We propose to identify delivery trips of a courier based on the following two rules: 1) a trip begins when the number of receiving parcels stops increasing, and the number of delivering parcels begins to increase; and 2) a trip ends if the opposite condition holds.



(a) A Normal Workday. (b) A Promotion Day.

Fig. 6. # of Waybills Received & Delivered w.r.t. Time of Day.

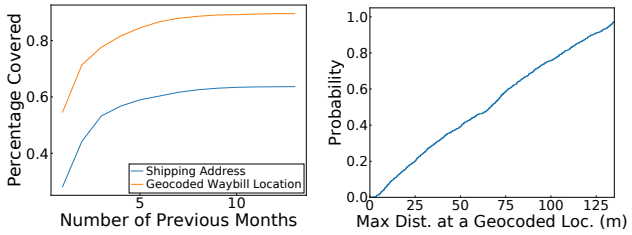
Based on the identified delivery trips, we separate waybills and stay points by the delivery trips that contain them.

4 Delivery Location Mining

In this component, we mine location mapping from the shipping address to the delivery location based on historical delivery caused stay points, so that we can employ the actual delivery location of each waybill to improve the accuracy of delivery time inference in the following component. We also generate a mapping from the Geocoded waybill location to the delivery location, so that delivery location knowledge can be generalized to future unseen addresses.

Motivation. The reason the delivery location knowledge mined from historical data can improve the delivery time inference is that addresses or Geocoded waybill locations in historical waybills often appear in the future. As shown in Figure 7(a), the shipping addresses and Geocoded waybill locations of waybills in the previous 4 months cover more than 57% addresses and 80% locations in the last month, respectively. In DTInf [1], we only infer the delivery location for each Geocoded waybill location because it is more general. However, we further find parcels whose shipping addresses sharing the same Geocoded location can be delivered at stay points that are far away from each other.

Figure 7(b) shows the distribution of the maximum distance differences of delivery caused stay points of waybills whose Geocoded location are the same. The large distance gaps indicate that there might be several delivery locations for a Geocoded location. Therefore, we mine both the address-based and the Geocoded location-based mappings, so that during the inference stage, we can not only obtain accurate delivery locations for seen addresses, but also share the knowledge to unseen ones whose Geocoded locations have previously appeared.

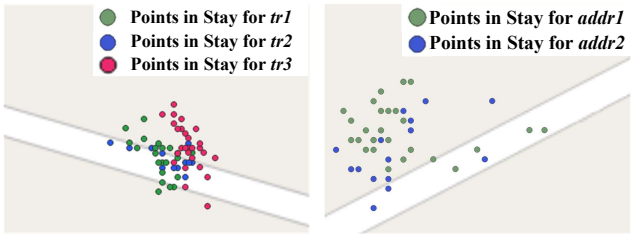


(a) Address/Location Overlap. (b) Max Distance of Stay Points.

Fig. 7. Motivations of Delivery Location Mining.

Main Idea. The delivery location mining mainly consists of three steps: 1) *Inverted Indexing*, which stores all historical delivery caused stay points for each shipping address or Geocoded waybill location; 2) *Location Inference*, which infers the raw delivery location; and 3) *Location Refinement*, that clusters the raw delivery locations which are spatially very close to generate the final delivery locations. This is inspired by two insights discovered in the dataset:

- *Multiple delivery caused stay points*: Figure 8(a) shows 3 delivery caused stay points of an address in different trips. It is noticeable that although those stay points are quite close, there are still minor differences. If all stay points of a shipping address/Geocoded location are leveraged, the delivery location inference can be more accurate.
- *Many-to-one mapping*: Given the fact that many waybills with different shipping addresses are delivered at the same delivery location, it is a many-to-one relationship between shipping addresses and delivery locations. Figure 8(b) shows delivery caused stay points of two shipping addresses. It can be noticed that their delivery caused stay points have considerably large overlaps, which indicates they potentially correspond to the same delivery location in the real world. Besides, this is also the case for some Geocoded waybill locations due to non-consistent results from different Geocoding services or centralized delivery spots of different buildings. If we just infer the delivery location for each shipping address/Geocoded waybill location individually, those delivery locations have high possibility to be distinct even though they are spatially very close, which further would degrade the effectiveness of delivery time inference as we demonstrated in the experiments.



(a) Multiple Stay Points. (b) Many-to-one Mapping.

Fig. 8. Insights of Delivery Location Mining.

Implementation. The procedure of the delivery location mining is presented in Algorithm 1, which takes historical trips TR and a distance threshold D, and returns two location mappings: $\mathcal{R}_a$, which is from the shipping address to the delivery location, and $\mathcal{R}_g$, which is from the Geocoded waybill location. We first build two inverted indexes $\mathcal{M}_a$ and $\mathcal{M}_g$ to store the delivery caused stay points for each shipping address $addr$ and each Geocoded waybill location l_a by iterating over all historical trips. In each trip tr, we iterate over waybills delivered during the trip, i.e., $tr.W$, and find the delivery caused stay point for each waybill w by querying the stay point whose temporal range includes the delivery time $w.t_d$ in trip stay points $tr.SP$, and add it to $\mathcal{M}_a$ and $\mathcal{M}_g$ (Line 2-6). Note that, we only store unique stay points for each $addr$ and l_a to avoid adding duplicated stay points given multiple waybills with one address/Geocoded waybill location for each trip. Then, for each index key $addr$ in $\mathcal{M}_a$, we infer the raw delivery location using the centroid of its delivery caused stay points, and store the location mapping to $\mathcal{A}_{addr}$ (Line 7-8). And for each index key l_a in $\mathcal{M}_g$, its delivery caused stay points might correspond to multiple delivery locations. Therefore, we first apply a hierarchical clustering algorithm [6] on all delivery caused stay points to form several clusters bounded by the given distance threshold D, and the clustering result $\mathcal{H}$ stores the mapping from the stay point to its cluster centroid. The clustering algorithm first treats each stay point as a cluster, and then iteratively merges two clusters with the minimum distance, which is defined as the geographical

Algorithm 1 Delivery Location Mining.

Input: The historical trips TR; the distance threshold D.
Output: The address-based mapping $\mathcal{R}_a$ and the Geocoded location-based mapping $\mathcal{R}_g$.

1: $\mathcal{M}_g, \mathcal{M}_a, \mathcal{A}_g, \mathcal{A}_a, \mathcal{R}_g, \mathcal{R}_a \leftarrow \emptyset$;
/* Inverted Indexing*/
2: **for** $tr \in TR$ **do**;
3: **for** $w \in tr.W$ **do**
4: $sp \leftarrow temporal\_query(tr.SP, w.t_d)$;
5: $\mathcal{M}_a[w.addr] \leftarrow \mathcal{M}_a[w.addr] \cup \{sp\}$;
6: $\mathcal{M}_g[w.l_a] \leftarrow \mathcal{M}_g[w.l_a] \cup \{sp\}$;
/* Location Inference */
7: **for** $addr \in \mathcal{M}_a$ **do**
8: $\mathcal{A}_a[addr] \leftarrow centroid\_calculation(\mathcal{M}_a[addr])$;
9: **for** $l_a \in \mathcal{M}_g$ **do**
10: $\mathcal{H} \leftarrow hierarchical\_clustering(\mathcal{M}_g[l_a], D)$;
11: $\mathcal{A}_g[l_a] \leftarrow main\_centroid\_selection(\mathcal{H})$;
/* Location Refinement */
12: $L \leftarrow \{\mathcal{A}_a[addr] | \forall addr \in \mathcal{A}_a\} \cup \{\mathcal{A}_g[l_a] | \forall l_a \in \mathcal{A}_g\}$;
13: $\mathcal{H} \leftarrow hierarchical\_clustering(L, D)$;
14: **for** $addr \in \mathcal{A}_a$ **do**
15: $\mathcal{R}_a[addr] \leftarrow \mathcal{H}[\mathcal{A}_a[addr]]$;
16: **for** $l_a \in \mathcal{A}_g$ **do**
17: $\mathcal{R}_g[l_a] \leftarrow \mathcal{H}[\mathcal{A}_g[l_a]]$;
18: **return** $\mathcal{R}_a$ and $\mathcal{R}_g$;

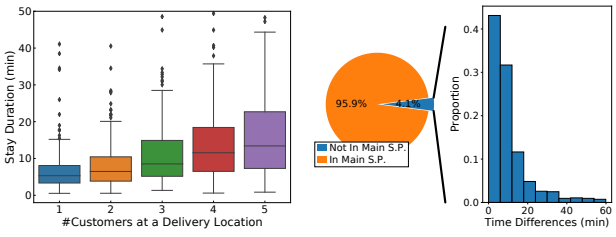
distance between two cluster centroids, to form a new cluster, until there does not exist two clusters whose distance is smaller than D. Next, we select the cluster centroid with the maximum number of stay points as the raw delivery location of the Geocoded waybill location, and store the location mapping to $\mathcal{A}_g$ (Line 9-11). After that, we apply the same clustering algorithm on all discovered raw delivery locations obtained from $\mathcal{A}_a$ and $\mathcal{A}_g$ (Line 12-13). Eventually, the final delivery location for each $addr$ and l_a is the centroid of the cluster formed by raw delivery locations. The final location mapping is stored in $\mathcal{R}_a$ and $\mathcal{R}_g$, and returned (Line 14-18).

5 Delivery Event-based Matching

In this component, we infer the delivery time for completed waybills based on stay points in the delivery trip. We denote the set of waybills to be delivered at the same delivery location in a trip as a *delivery event*, which can be obtained based on the previously mined location mappings. For each trip, we first construct several delivery events, then select the best-matched stay point for each event using an attention-based selection model. The time of the selected stay point is further treated as the delivery time of waybills in the event.

The reasons to perform the delivery event-level matching for waybills in a trip are two-fold:

- *Correlations between delivery events and stay points*: The delivery event will affect the characteristics of the delivery caused stay point. The box plot in Figure 9(a) shows the duration distribution w.r.t. the number of customers of a delivery event ($D = 20$m). It is obvious that a courier would stay longer if he/she needs to deliver for more customers. Such characteristics cannot be captured if we perform the inference task for each waybill individually.
- *Location by location delivery*: A courier usually continuously delivers all parcels at the same delivery location, e.g., a residential building. For a delivery location in each historical trip, we can find a delivery caused stay point, during the time interval of which the majority of waybills at that location are completed. We call such stay point as the *main (delivery caused) stay point* of that delivery event. The pie chart in Figure 9(b) shows that 95.9% waybills are completed during the time intervals of the main stay points. Though 4.1% waybills are completed out of them, their delivery time differences w.r.t. the main stay points are small as the histogram shows. Therefore, if we correctly infer the main stay point of each delivery event, the time inference errors for waybills are acceptable.



(a) Duration w.r.t #Customers. (b) Delivered in/out Main S.P..

Fig. 9. Insights of Delivery Event Construction.

5.1 Delivery Event Construction

In this step, we group completed waybills according to their delivery locations to form several delivery events based on previously mined location mappings.

In order to construct delivery events, we first need to obtain the delivery location for each waybill based on its shipping address. For historical waybills, the delivery locations can be directly obtained using the address-based mapping. However, during the online inference phase, it is possible that some shipping addresses are never seen. Recall that in Section 4, we maintain the address-based mapping and the Geocoded location-based mapping. When running online, there is an accuracy-generalization trade-off: while the delivery location knowledge from address-based mapping is more reliable, since each address usually corresponds to one actual delivery location (, which might not be true for each Geocoded waybill location), the possibility that an address has been seen before is smaller than that of a Geocoded waybill location. Therefore, we propose a two-level correction strategy (first address-level, then Geocoding-level) which follows the following priorities to obtain the delivery location of a waybill: 1) We first obtain the delivery location by using its address as the key to query the address-based mapping $\mathcal{R}_a$ mined from the historical data. 2) If the address has never appeared, we use the Geocoding-based mapping $\mathcal{R}_g$ to obtain the delivery location based on its Geocoded location. 3) If neither its address nor its Geocoded location has appeared in the historical data, we directly use its Geocoded location as the delivery location.

The procedure of the delivery event construction is detailed in Algorithm 2, which takes completed waybills W, and two location mappings $\mathcal{R}_a$, $\mathcal{R}_g$ as input, and returns a mapping $\mathcal{E}$ from delivery location to a set of waybills. At first, $\mathcal{E}$ is initialized as empty (Line 1). Then, we iterate over each waybill w in W, and add it to $\mathcal{E}$ according to the delivery location l_d derived from the shipping address. If the address appears in $\mathcal{R}_a$, we use $\mathcal{R}_a[w.addr]$ as l_d; if that is not the case, we check whether its Geocoding result l_a appears in $\mathcal{R}_g$ and can be used. If l_a is not in $\mathcal{R}_g$ either, we directly use l_a as l_d (Line 3-8). Next, we add w to the set of waybills sharing the same delivery location l_d (Line 9-12). After W has been iterated, $\mathcal{E}$ is returned (Line 13).

Algorithm 2 Delivery Event Construction.

Input: The completed waybills W, the address-based mapping $\mathcal{R}_a$, and the Geocoded location-based mapping $\mathcal{R}_g$.
Output: Delivery events $\mathcal{E}$.

1: $\mathcal{E} \leftarrow \emptyset$;
2: **for** $w \in W$ **do**
3: **if** $w.addr \in \mathcal{R}_a$ **then** ▷ Address-level Correction
4: $l_d \leftarrow \mathcal{R}_a[w.addr]$;
5: **else if** $w.l_a \in \mathcal{R}_g$ **then** ▷ Geocoding-level Correction
6: $l_d \leftarrow \mathcal{R}_g[w.l_a]$;
7: **else**
8: $l_d \leftarrow w.l_a$;
9: **if** l_d **not in** $\mathcal{E}$ **then**
10: $\mathcal{E}[l_d] \leftarrow \{w\}$;
11: **else**
12: $\mathcal{E}[l_d] \leftarrow \mathcal{E}[l_d] \cup \{w\}$;
13: **return** $\mathcal{E}$;

5.2 Stay Point Selection

After we construct several delivery events in a trip, we issue a spatial range query to find stay points within a certain radius for each delivery event, and then our task is to select a stay point, which corresponds to the main stay point of that delivery event. After the stay point is selected for each delivery event, the time of stay point is used as the inferred delivery time for all waybills in the delivery event.

In DTInf [1], we train an MLP model to classify whether a stay point is the main delivery caused stay point of an event, and during the inference phase, we select for each delivery event the stay point with the highest probability as the prediction result. Such a method independently classifies each stay point, while the existence of other stay points in the neighborhood of a delivery event would also affect the judgement about whether the stay point is caused by its delivery. For example, according to the domain knowledge, a courier is likely to deliver a parcel when he/she visits its delivery location for the first time in a trip, which means the stay point which has earlier time among candidates has higher possibility to be the delivery caused stay point. For another example, we find the duration of the delivery caused stay point is usually longer than other candidates, which might be accidentally generated. The ranking of duration is also a relative relationship.

In order to capture such temporal dependency and the inter-relationship among candidate stay points of a delivery event, as well as handle varying number of candidate stay points of different delivery events, we propose SPSelector, which is inspired by the pointer network [4]. The pointer network uses the RNN and the attention mechanism to learn the conditional probability of an output sequence with elements that are discrete tokens corresponding to positions in an input sequence.

Model Architecture. The overall architecture of SPSelector is shown in Figure 10. Our model is an end-to-end deep neural network based on the encoder-decoder architecture. It takes a delivery event and its candidate stay points as input, and generates the probability distribution among candidates. As shown in Figure 10, SPSelector consists of three layers: representation layer, encoder layer and selection layer. The representation layer utilizes various features extracted from the delivery event and candidate stay points to generate dense representations. The encoder layer uses the bidirectional Gated Recurrent Unit (GRU) [7] to incorporate the temporal dependency, fuse inter-relationship among candidates, and generates stay point embeddings. The selection layer uses the attention mechanism to produce the probability distribution among candidates by treating the dense representation of the delivery event as the query vector and stay point embeddings as the key vectors. Finally, the stay point with the highest probability is selected.

Representation Layer. The goal of the representation layer is to generate comprehensive representation from both delivery event and its candidate stay points.

- *Delivery event representation.* For each delivery event, we generate three types of features: 1) the aggregated waybill information, including the number of waybills, the number of customers, total weight, and total volume of waybills in the delivery event; 2) the POI category of the delivery location; and 3) the spatial context, which is formed by dividing the neighborhood of delivery location into uniform grids, and the aggregated waybill information in each grid are calculated. The first two types of features are both related to the characteristics of main delivery caused stay point (e.g., the delivery duration), and the third feature would make the model aware whether there are other delivery events in a certain neighborhood region to influence the appearance of stay point at that region. Given that the POI category is categorical, and the spatial context is sparse, to fuse those three types of features, we first feed the POI category into the embedding layer, and the spatial context into a dense layer to generate low-dimensional vectors, then they are concatenated with the aggregated waybill information to generate the delivery event representation, which also serves as a query vector in the selection layer. We denote it as $\mathbf{q} \in \mathbb{R}^m$.
- *Stay point representation.* For each stay point, we also generate three types of features: 1) the stay point characteristics, including the duration and area of the stay point; 2) the matching features, consisting of the distance and the bearing angle to the delivery location; and 3) the spatial context, which is obtained in the same way like the spatial context of the delivery event, except that the center is moved to the stay point. The first type of features is important to judge whether a stay point is a delivery caused stay point, while the second and the third type of features are vital to judge where the stay point is the delivery caused stay point of the *current* event. To fuse those features, similarly, we first feed the spatial context into another dense layer to generate a low-dimensional vector, then it is concatenated with the other features to generate the stay point representation.

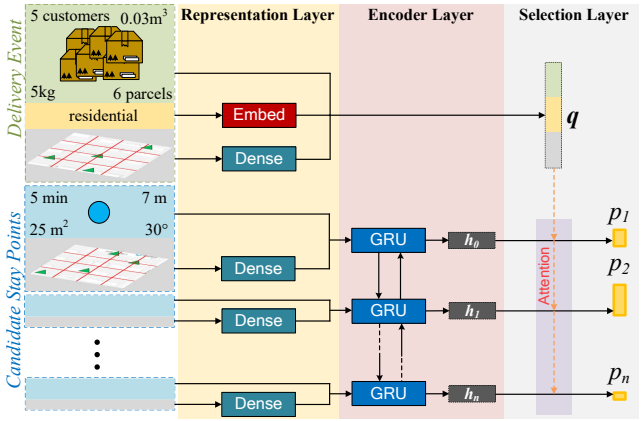


Fig. 10. SPSelector Architecture.

Encoder Layer. The encoder layer is designed to incorporate the temporal dependency and fuse the inter-relationship among stay points. To achieve this goal, we employ the bidirectional GRU [7]. The GRU is first proposed to solve the machine translation task, and the bidirectional extension allows two neural network layers to receive information from both past and future states by connecting them to a single output and ultimately improves the performance of some tasks, e.g., sequence classification. Since we not only want the module to capture the temporal dependency, but

also fuse information among candidates, the bidirectional GRU is suitable for our task.

We sequentially feed the stay point representations generated in the previous layer into the bidirectional GRU by the time order of the stay point. We name the output of the GRU as stay point embeddings, denoted as $\{\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_n\}$, where n is the number of candidates, $\mathbf{h}_i \in \mathbb{R}^{2h}$, and h is the number of hidden units of the GRU from one direction.

Selection Layer. The selection layer takes the stay point embeddings and the query vector, and produces the probability distribution among stay points.

Given the k-th stay point embedding in candidates, we employ an attention mechanism to adaptively calculate the matching score e_k between the query vector $\mathbf{q}$ and the stay point embedding $\mathbf{h}_k$:

$$e_k = \mathbf{v}^\top \tanh(\mathbf{W}\mathbf{h}_k + \mathbf{U}\mathbf{q} + \mathbf{b}) \tag{3}$$

where $\mathbf{v}, \mathbf{b} \in \mathbb{R}^p$, $\mathbf{W} \in \mathbb{R}^{p \times 2h}$, and $\mathbf{U} \in \mathbb{R}^{p \times m}$ are the parameters to be learned, and p is a hyper-parameter.

Then, e_k is normalized among all candidates by the softmax function to generate the probability p_k:

$$p_k = \frac{\exp(e_k)}{\sum_{i=1}^{n} \exp(e_i)} \tag{4}$$

Training and Prediction. In the training process, we consider the stay point selection as a classification problem with multiple classes. Therefore, the cross-entropy is used to calculate the selection loss, which is defined as follows:

$$\mathcal{L}(\theta) = -\sum_{i=1}^{n} y_i \log p_i \tag{5}$$

where θ are all learnable parameters, n is the number of candidate stay points of the event, y_i is a binary ground truth indicating whether the i-th candidate of the event is the main delivery caused stay point of it.

In the inference process, the stay point with the maximum predicted probability is selected, and the time of stay point is used as the inferred delivery time for all waybills in the delivery event.

6 Experiments

6.1 Experimental Settings

Datasets. The datasets contain trajectories and waybills of 5 couriers who are known to confirm the delivery roughly on time at a delivery station in Tongzhou District, Beijing over a period of about 15 months (from Apr. 12$^{\text{nd}}$, 2018 to Jul. 7$^{\text{th}}$, 2019).

- **Couriers' trajectories.** They are raw GPS logs generated by couriers' PDAs, where each record contains a courier ID, a location, and a timestamp. The average sampling time interval is 7.4 seconds. The datasets contain 5.93 million GPS points.
- **Waybills.** Each record contains a customer ID, a courier ID, parcel information (e.g., weight and volume), the time when the parcel is received, the time when the parcel is delivered, and a Geocoded waybill location. The datasets contain 274 thousand waybills.

After the data pre-processing step, waybills and stay points detected from trajectories are organized by delivery trips. The trips without trajectories are dropped. There are 3,653 delivery trips in total. For each courier, we use his/her former 80% trips for training, the following 10% trips for validating, and the last 10% trips for testing. The delivery location mining is conducted based on the training and validation trips, which contain 2,506 unique Geocoded waybill locations. 12.9% Geocoded waybill locations in the test trips have not appeared in former trips.

Evaluation Metrics. We use the time of delivery caused stay points for waybills as the ground-truth. The reason is that the delivery time recorded in waybills is not completely accurate. According to some domain experts, a courier might confirm the delivery of a parcel at any time during the delivery for the corresponding delivery location. However, the recorded delivery time can indicate the time period of the stay point at which the parcel is delivered. Therefore, we report the MAE and the RMSE based on the inferred delivery time and the time of delivery caused stay points to mitigate the error of human annotation as much as possible. We also use the accuracy, which is defined as the proportion of waybills whose corresponding delivery caused stay points are correctly selected (i.e., their inferred delivery times are accurate).

Baselines. To the best of our knowledge, there is no existing solution that can exactly tackle our problem. Therefore, we design the following four baselines for comparison:

- Random Inference (RDInf): We randomly select a stay point from candidates as its delivery caused stay point.
- Spatial Nearest Inference (SNInf): SNInf matches each waybill with its closest stay point in candidates.
- Temporal Longest Inference (TLInf): TLInf selects the stay point in candidates with the longest duration.
- Temporal Earliest Inference (TEInf): TEInf selects the stay point in candidates with the earliest time.

Variants. We also compare DTInf$^+$ with its seven variants to evaluate the effectiveness of each component:

- DTInf [1]: It is the prior version of this work. The main difference between DTInf and DTInf$^+$ is that DTInf assumes each Geocoded waybill location corresponds to one delivery location, and classifies each stay point independently using an MLP, which does not consider the interaction relationship among candidate stay points, neither the spatial context of stay points and delivery events.
- DTInf$^+$-nPN: This variant doesn't use the pointer network structure, and trains an MLP model which takes features of a delivery event and a stay point as input, and predicts whether the stay point is the delivery caused stay point of that event.
- DTInf$^+$-nC: This variant does not correct the delivery locations, which treats the Geocoded waybill location of a waybill as its delivery location.
- DTInf$^+$-AC: This variant obtains the delivery location for each waybill only based on the address-based mapping. If its address is not in the mapping, its Geocoded waybill location is treated as the delivery location.
- DTInf$^+$-nM: This variant performs the stay point selection without a model, and selects the closest stay point to the corrected delivery location.

- DTInf$^+$-nCTX: This variant does not use the spatial context features.
- DTInf$^+$-nE: This variant does not construct delivery events. Instead, it infers delivery caused stay point for each waybill based on the same model, but the delivery event features are replaced with individual waybill features.

Parameter Settings. There are 16 POI categories we obtained via the reverse Geocoding service. The location merging parameter D is set to $20m$ by default. The query radius R is set to $70m$ to find stay point candidates to cover all delivery caused stay points according to Figure 2(a). For the spatial context feature, we define the $140m \times 140m$ square region as the neighborhood, and divide the region into 3×3 uniform grids, which shows the best performance. In SPSelector, $h = 16$ and $p = 32$. During the training phase, we leverage Adam [8] to perform network training with a learning rate $1e$-4 and the batch size is 16. The learning rate is halved every 20 epochs.

Implementations. Our algorithms are implemented in Python. SPSelector is implemented by PyTorch. Experiments are conducted on a workstation with an Intel(R) Core(TM) CPU i7-8700K @ 3.7GHz, 32GB memory, and Windows 10.

6.2 Data Descriptions

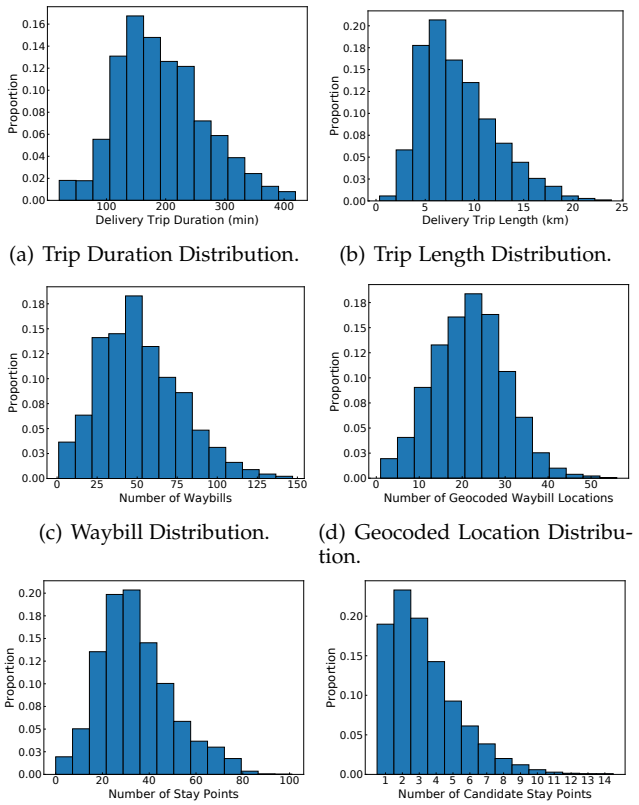
Delivery Trip Distribution. Figure 11(a) and 11(a) gives the distribution of the trip duration and the trip length detected from the delivery trip identification step in Section 3.3. The average duration of a delivery trip is about 3.2 hours, and the maximal duration does not exceed 7 hours, because a courier needs to go back to the station to receive newly arrived parcels after a certain time period. The average length is 8.3 km. Since a courier is assigned to deliver parcels for some regions that are spatially close, the length of the trip usually is not long.

Waybill Distribution. Figure 11(c) and 11(d) show the distribution of the number of waybills and unique Geocoded waybill locations in each delivery trip, respectively. As can be observed, a courier needs to deliver 52 waybills in each delivery trip on average. Since some waybills are in the same building, or belong to the same customer, there are 22 unique Geocoded locations to be delivered on average.

Stay Point Distribution. Figure 11(e) shows the distribution of the number of stay points in each delivery trip. The average number of stay points is 34, which is larger than the average number of Geocoded waybill locations. It validates our claim that a courier stays at a location during a delivery trip not only because of the delivery, but also for some other purposes. We also plot the distribution of the number of stay points near each waybill in Figure 11(f). As shown in the figure, 81% waybills contain more than one stay point in their neighborhood ($R = 70$m), which implies that the stay point selection is necessary.

6.3 Effectiveness Evaluation

Overall Evaluation. The overall performance of DTInf$^+$ compared with baselines and variants is shown in Table 1. Among 4 baselines, RDInf only achieves 41.0% accuracy. TEInf is better than random guess, which means the earlier appeared stay point is more likely to be the delivery caused



(a) Trip Duration Distribution. (b) Trip Length Distribution.

(c) Waybill Distribution. (d) Geocoded Location Distribution.

(e) Trip Stay Point Distribution. (f) Candidate Stay Point Distribution.

Fig. 11. Dataset Descriptions.

stay point. Comparing SNInf and TLInf, we can find that the matching based on the duration is a better heuristic. DTInf$^+$ outperforms the best baseline (TLInf) by 52.8% in terms of MAE. After improving the delivery location mining methods and modeling techniques, we achieve 30.8% performance gain in terms of MAE, compared to DTInf.

Different Delivery Location Correction Methods. We also compare different delivery location correction methods in Table 1 to show the importance of delivery location mining. Comparing SNInf and DTInf$^+$-nM, we can see that after the locations are corrected, a 30.7% performance gain is witnessed using the spatial nearest heuristic. Comparing DTInf$^+$-nC and DTInf$^+$, we find that if the model is trained based on the Geocoded locations, its effectiveness is degraded. DTInf$^+$-AC is also less effective than DTInf$^+$, which shows the necessity of maintaining both the address-based mapping and the Geocoded location-based mapping. It is inevitable that during the online inference, new addresses would appear. The Geocoded location-based mapping provides an effective backup in case an address is new while its Geocoded location has ever appeared in the historical data. We also note that if express lockers exist in the delivery region, the performance of non-correction-based methods (i.e., all baselines and DTInf$^+$-nC) could be even worse since the delivery caused stay point might be far away from the Geocoded waybill location, while other methods are less affected, which consider the distance to the actual delivery location.

Different Stay Point Selection Strategies. We also compare different stay point selection strategies in Table 1 to show the

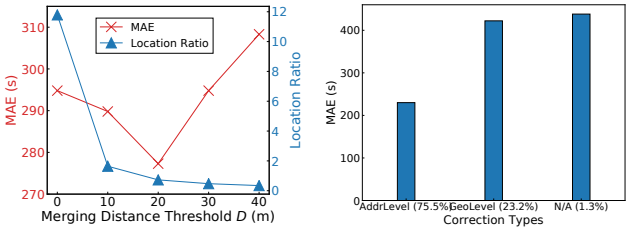
TABLE 1
Comparison with Baselines and Variants.

Methods	Accuracy (%)	RMSE (s)	MAE (s)
RDInf	41.0	2725.8	1254.5
TEInf	47.6	2434.7	1124.5
SNInf	55.8	2361.5	868.9
TLInf	71.3	1713.4	588.1
DTInf$^+$-nM	66.7	1954.5	601.8
DTInf	75.5	1365.6	401.0
DTInf$^+$-nPN	77.4	1278.1	336.6
DTInf$^+$-nC	78.1	1324.9	376.4
DTInf$^+$-AC	78.4	1154.0	316.5
DTInf$^+$-nE	79.5	1183.9	312.6
DTInf$^+$-nCTX	79.5	1082.5	286.5
DTInf$^+$ (ours)	**79.8**	**1051.7**	**277.3**

advantages of the proposed method. Comparing DTInf$^+$-nM and DTInf$^+$, we find the modeling-based stay point selection significantly outperforms simple heuristics. The performance of DTInf$^+$-nPN is much worse than DTInf$^+$, which shows the importance of considering all candidates jointly rather than independently classifying each of them, since the pointer network-like structure is able to capture the sequence and relative information among candidate stay points. The performance gap between DTInf$^+$ and DTInf$^+$-nE shows the superiority of the delivery event modeling than modeling each waybill individually. The reason is that when a courier stays at a location for a while, he/she usually delivers a batch of parcels, all of which contribute to the duration of the stay. Therefore, the event-based matching is more reasonable and thus has better modeling performance. DTInf$^+$-nCTX is also worse, which shows the spatial context of the delivery event and the stay point also affect the judgement of delivery caused stay points.

Merging Distance Selection. In order to model the delivery event accurately, we need to select an appropriate location merging parameter D by varying it from 0m (no merging) to 40m. The time inference error is reported in Figure 12(a), which shows the MAE first drops and then increases. The reason is that when D becomes larger, redundant delivery locations are merged, which makes the delivery event modeling more accurate. However, when D is larger than 20m, the performance is degraded, because we might merge adjacent delivery locations by mistake. We also report the ratio between the number of delivery locations and Geocoded waybill locations (denoted as the location ratio) in the same figure. It shows that the location ratio first decreases sharply, and then changes smoothly, which also demonstrates the redundancy issue of delivery locations. Therefore, we set the turning point 20m as the merging distance D. We also plot the distribution of the distance shifts of correction locations with respect to the delivery caused stay points in Figure 2(a).

Different Correction Types. We are also interested in the performance differences of DTInf$^+$ when faced with waybills whose address/Geocoded locations have ever been seen or not. Figure 12(b) shows that if we can obtain the delivery locations based on the addresses (AddrLevel), which takes up for 75.5%, the MAE is 230s. If the Geocoded waybill locations have appeared while the addresses have not (GeoLevel), which takes up for 23.2%, the error is 422s. For waybills whose addresses or Geocoded waybill



(a) Different Merging Distance. (b) Different Correction Types.
Fig. 12. Effectiveness Experiments.

locations have never appeared in history (N/A), the error is 438s. Although, if the delivery location is obtained from the Geocoded location-based mapping, the performance improvement compared with directly treating the Geocoded location as the delivery location is not significant, waybills whose addresses are new, but their Geocoded locations have ever appeared take up for 23.2% in our evaluation dataset. Two-level correction ultimately makes the overall inference error smaller according to DTInf$^+$-AC and DTInf$^+$ in Table 1. Another interesting point is that although there are 12.9% Geocoded waybill locations have not appeared in history as we mentioned in the datasets, those waybills only correspond to 1.3% in total test waybills, which indicates that the locations that more waybills are affiliated with, have a higher chance to appear in history, and the delivery location correction is applicable to the majority of waybills.

6.4 Case Study

We further give a case study of a delivery trip on the morning of Jun. 16th, 2019, which is one of the delivery trips in the evaluation dataset. Figure 13 shows the satellite image of a region in Tongzhou District. There is a waybill whose Geocoded location is displayed with the red triangle. The blue dots are the centroids of stay points detected from the courier's trajectory of the corresponding delivery trip, where the stay point with the longest duration in the neighborhood is sp_4. However, according to the ground-truths, the parcel is delivered during the time interval of sp_3, which leads to a great time inference error. Fortunately, this Geocoded waybill location has been delivered multiple times in history, so we are able to correct it. The corrected location is shown with the green triangle, which is much closer to the delivery caused stay point. Nevertheless, if we just employ the spatial nearest heuristic, sp_{18} would be inferred as the matched stay point, which also leads to

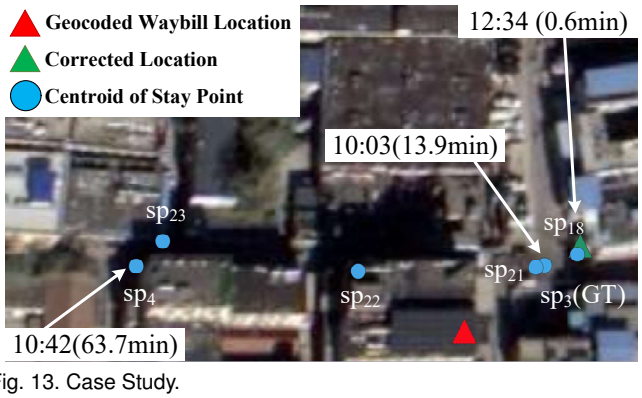


Fig. 13. Case Study.

large inference error. DTInf$^+$ successfully selects sp_3 among candidate stay points because it considers various factors.

7 System Deployment

Our delivery time inference system is deployed internally in JD Logistics. In order to process massive couriers' trajectories, we leverage our self-developed platform, JD Urban Spatio-Temporal Data Engine (JUST) [9], to efficiently perform the noise filtering and the stay point detection in the distributed environment based on Apache Spark and HBase. The spatio-temporal index is also built over detected stay points based on JUST to accelerate the process of querying stay points near the delivery locations. When running online, our inference process is activated as soon as a courier ends his/her trips and commits parcels he/she fails to deliver. For waybills whose Geocoded waybill locations newly appeared, couriers would be asked to record the delivery time optionally in order to correct the delivery locations for better delivery time inference in the future.

The interface of our system is shown in Figure 14, which allows operators to visualize couriers' delivery trips and understand the inference process. The interface contains four components:

Operation View. In this view, the operator can perform several operations. There are four main buttons: 1) *Retrieve*, which is used to query waybills and trajectories of a trip that is specified; 2) *Correct*, which corrects the delivery location of waybills in the current trip; 3) *Query*, that issues spatio-temporal query to find stay points near the delivery location of each waybill; and 4) *Infer*, that infers the delivery caused stay point for each waybill.

Main Map View. The right part is the main map view. When the delivery trip is retrieved, courier's trajectories (grey line), stay point centroids (blue circle), and Geocoded waybill locations (red triangle) are visualized. After *Correct* is clicked, the corrected delivery locations are displayed with the green triangles, and the big red circles indicate the querying neighborhood. Finally, when the operator clicks *Infer*, a link would be generated between the corrected location and the inferred delivery caused stay point.

Waybill Information View. This view displays the detailed information about the retrieved waybills. The waybill ID, the customer ID, the weight and the volume are shown. If one of the waybills is selected, the location of the waybill would appear within the map view, and the inferred stay point will appear in the result view.

Result View. The view shows the detailed information of the inferred delivery caused stay point for the selected waybill. It displays the start time, the end time, and the duration of the stay point.

Fig. 14. System Interface.

8 Related Work

Trajectory Annotation. In this work, we essentially want to annotate some stay points in trajectories with parcels delivered during the time interval of them, which is related to the trajectory annotation. The trajectory annotation aims to enrich trajectories with the semantic information [10]. The annotation techniques are mainly concerned with annotating trajectories with maps [11], [12], [13], [14], [15], [16] and recognizing the transportation modes [17], [18]. Annotating moving trajectories with roads is also known as the map matching [11], [12]. Annotating stay points with POIs is usually based on the geometric intersection or the spatial nearest neighbor [13], [14]. Many candidate POIs may exist in some densely populated urban areas, therefore, Yan et al. [15] attempted to infer the POI categories based on a Hidden Markov Model to maximize the visiting sequential probability. Keles et al. [19] predicted POI categories using a Bayesian Network, which considers the time of the day, the day of the week, and the duration of the stay point. Suzuki et al. [16] inferred the exact POIs a user visited under the integer linear programming framework, which also considers various features from POIs and stay points. For the POI assignment tasks, the stay duration differences are usually caused by different POI categories, while in our problem, the number of customers at a delivery location plays the dominant role. Apart from that, for the trajectory annotation, a specific POI would be annotated for multiple times or not be assigned, both of which are not acceptable in our scenario.

Trajectory Data Mining. The trajectory data mining [5] studies discovering various knowledge from massive trajectory data. To enhance the existing maps, [20], [21], [22], [23] studied the road network generation or refinement based on crowd sourced trajectories, and [24], [25], [26], [27] studied discovering interesting places from trajectory hotspots. To help urban planning, [28], [29] gave the bike path lane planning or electric fence construction recommendation. To increase the commercial profits, [30], [31] aimed to select the best location for the billboard placement. To improve the user experience, [26], [32] studied the traveling recommendation. In this work, we discover the delivery locations based on trajectories and the recorded delivery time, which are further used to help the delivery caused stay point recognition.

Urban Computing. Urban computing [33] aims to solve the issues caused by human's rapid progress in urbanization, such as anomaly detection [34], [35], crime rate inference [36], air quality prediction [37], and resource rebalancing [38], [39]. In our work, we focus on easing the burden of couriers by automatically inferring the delivery time based on their trajectories.

9 Conclusion

In this paper, we propose DTInf$^+$, a system to automatically fill the delivery time based on couriers' trajectories. Our method first separates waybills and stay points detected from trajectories by delivery trips, then mines delivery locations knowledge from historical trips by maintaining address-based mapping and Geocoded location-based mapping, and finally during the online inference, it constructs delivery events for waybills based on offline mined delivery locations, and predicts the delivery caused stay point for each event using a pointer network-like model SPSelector, which is further used to infer the delivery time of waybills in it. Experiments show our method significantly outperforms baselines by at least 52.8%. And a case study is further conducted to illustrate the advantage of our solution. Finally, a system is deployed in JD Logistics. In this work, the delivery location mining is only applicable to couriers who confirm the delivery of parcels roughly on time in history. In the future, we would explore ways to infer the delivery locations for addresses even if couriers confirm deliveries with significant delays, so that the automatic delivery time filling can also be applied to those couriers.

Acknowledgments

This work was supported by the National Key R&D Program of China (2019YFB2101805), the NSFC Grant (61976168, 62076191), and Beijing Academy of Artificial Intelligence (BAAI). This work was also supported by the Nanyang Technological University Start-UP Grant from the College of Engineering under Grant M4082302 and by the Ministry of Education, Singapore, under its Academic Research Fund Tier 1 (RG20/19 (S)).

References

[1] S. Ruan, Z. Xiong, C. Long, Y. Chen, J. Bao, T. He, R. Li, S. Wu, Z. Jiang, and Y. Zheng, "Doing in one go: delivery time inference based on couriers' trajectories," in *KDD*, 2020.

[2] Q. Li, Y. Zheng, X. Xie, Y. Chen, W. Liu, and W.-Y. Ma, "Mining user similarity based on location history," in *SIGSPATIAL*. ACM, 2008, p. 34.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT, 2016.

[4] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *NIPS*. Curran Associates, Inc., 2015, pp. 2692–2700.

[5] Y. Zheng, "Trajectory data mining: an overview," *TIST*, vol. 6, no. 3, p. 29, 2015.

[6] J. H. Ward Jr, "Hierarchical grouping to optimize an objective function," *Journal of the American statistical association*, vol. 58, no. 301, pp. 236–244, 1963.

[7] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[8] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[9] R. Li, H. He, R. Wang, Y. Huang, J. Liu, S. Ruan, T. He, J. Bao, and Y. Zheng, "Just: Jd urban spatio-temporal data engine," in *ICDE*. IEEE, 2020.

[10] B. H. Albanna, I. F. Moawad, S. M. Moussa, and M. A. Sakr, "Semantic trajectories: a survey from modeling to application," in *IF&GIS*. Springer, 2015, pp. 59–76.

[11] J. Yuan, Y. Zheng, C. Zhang, X. Xie, and G.-Z. Sun, "An interactive-voting based map matching algorithm," in *MDM*. IEEE, 2010, pp. 43–52.

[12] P. Newson and J. Krumm, "Hidden markov map matching through noise and sparseness," in *SIGSPATIAL*, 2009, pp. 336–343.

[13] L. O. Alvares, V. Bogorny, B. Kuijpers, J. A. F. de Macedo, B. Moelans, and A. Vaisman, "A model for enriching trajectories with semantic geographical information," in *Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems*, 2007, pp. 1–8.

[14] D.-W. Choi, J. Pei, and T. Heinis, "Efficient mining of regional movement patterns in semantic trajectories," *VLDB*, vol. 10, no. 13, pp. 2073–2084, 2017.

[15] Z. Yan, D. Chakraborty, C. Parent, S. Spaccapietra, and K. Aberer, "Semantic trajectories: Mobility data computation and annotation," *TIST*, vol. 4, no. 3, pp. 1–38, 2013.

[16] J. Suzuki, Y. Suhara, H. Toda, and K. Nishida, "Personalized visited-poi assignment to individual raw gps trajectories," *TSAS*, vol. 5, no. 3, pp. 1–28, 2019.

[17] Y. Zheng, L. Liu, L. Wang, and X. Xie, "Learning transportation mode from raw gps data for geographic applications on the web," in *WWW*, 2008, pp. 247–256.

[18] S. Dabiri and K. Heaslip, "Inferring transportation modes from gps trajectories using a convolutional neural network," *Transportation research part C: emerging technologies*, vol. 86, pp. 360–371, 2018.

[19] I. Keles, M. Schubert, P. Kröger, S. Šaltenis, and C. S. Jensen, "Extracting visited points of interest from vehicle trajectories," in *Proceedings of the Fourth International ACM Workshop on Managing and Mining Enriched Geo-Spatial Data*, 2017, pp. 1–6.

[20] S. Wang, Y. Wang, and Y. Li, "Efficient map reconstruction and augmentation via topological methods," in *SIGSPATIAL*. ACM, 2015, p. 25.

[21] S. He, F. Bastani, S. Abbar, M. Alizadeh, H. Balakrishnan, S. Chawla, and S. Madden, "Roadrunner: improving the precision of road network inference from gps trajectories," in *SIGSPATIAL*, 2018, pp. 3–12.

[22] S. Ruan, C. Long, J. Bao, C. Li, Z. Yu, R. Li, Y. Liang, T. He, and Y. Zheng, "Learning to generate maps from trajectories," in *AAAI*, vol. 34, no. 01, 2020, pp. 890–897.

[23] L. Zhao, J. Mao, M. Pu, G. Liu, C. Jin, W. Qian, A. Zhou, X. Wen, R. Hu, and H. Chai, "Automatic calibration of road intersection topology using trajectories," in *ICDE*. IEEE, 2020, pp. 1633–1644.

[24] D. Ashbrook and T. Starner, "Learning significant locations and predicting user movement with gps," in *Proceedings. Sixth International Symposium on Wearable Computers*. IEEE, 2002, pp. 101–108.

[25] C. Zhou, D. Frankowski, P. Ludford, S. Shekhar, and L. Terveen, "Discovering personal gazetteers: an interactive clustering approach," in *Proceedings of the 12th annual ACM international workshop on Geographic information systems*, 2004, pp. 266–273.

[26] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma, "Mining interesting locations and travel sequences from gps trajectories," in *WWW*, 2009, pp. 791–800.

[27] V. W. Zheng, Y. Zheng, X. Xie, and Q. Yang, "Collaborative location and activity recommendations with gps history data," in *WWW*, 2010, pp. 1029–1038.

[28] J. Bao, T. He, S. Ruan, Y. Li, and Y. Zheng, "Planning bike lanes based on sharing-bikes' trajectories," in *KDD*, 2017, p. 1377–1386.

[29] Y. Zhang, D. Lin, and Z. Mi, "Electric fence planning for dockless bike-sharing services," *Journal of cleaner production*, vol. 206, pp. 383–393, 2019.

[30] Y. Li, J. Bao, Y. Li, Y. Wu, Z. Gong, and Y. Zheng, "Mining the most influential k-location set from massive trajectories," in *SIGSPATIAL*, 2016, pp. 1–4.

[31] P. Zhang, Z. Bao, Y. Li, G. Li, Y. Zhang, and Z. Peng, "Trajectory-driven influential billboard placement," in *KDD*, 2018, pp. 2748–2757.

[32] S. Jiang, X. Qian, T. Mei, and Y. Fu, "Personalized travel sequence recommendation on multi-source big social media," *TBD*, vol. 2, no. 1, pp. 43–56, 2016.

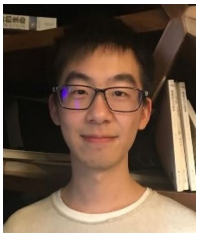
[33] Y. Zheng, L. Capra, O. Wolfson, and H. Yang, "Urban computing: concepts, methodologies, and applications," *TIST*, vol. 5, no. 3, p. 38, 2014.

[34] H. Zhang, Y. Zheng, and Y. Yu, "Detecting urban anomalies using multiple spatio-temporal data sources," *IMWUT*, vol. 2, no. 1, pp. 1–18, 2018.

[35] T. Fuse and K. Kamiya, "Statistical anomaly detection in human dynamics monitoring using a hierarchical dirichlet process hidden markov model," *TITS*, vol. 18, no. 11, pp. 3083–3092, 2017.

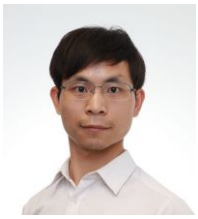
[36] H. Wang, D. Kifer, C. Graif, and Z. Li, "Crime rate inference with big data," in *KDD*, 2016, pp. 635–644.

[37] X. Yi, J. Zhang, Z. Wang, T. Li, and Y. Zheng, "Deep distributed fusion network for air quality prediction," in *KDD*, 2018, pp. 965–973.
[38] S. Ji, Y. Zheng, Z. Wang, and T. Li, "A deep reinforcement learning-enabled dynamic redeployment system for mobile ambulances," *IMWUT*, vol. 3, no. 1, pp. 1–20, 2019.
[39] J. Liu, L. Sun, W. Chen, and H. Xiong, "Rebalancing bike sharing systems: A multi-source data smart optimization," in *KDD*, 2016, pp. 1005–1014.



Sijie Ruan is a Ph.D. candidate at the School of Computer Science and Technology, Xidian University. He received his B.E. degree from Xidian University in 2017. His research interests include Trajectory Data Mining, Urban Computing, and Spatio-temporal Data Management. He was a research intern at Microsoft Research Asia from 2016 to 2017, and an algorithm engineer intern at JD Intelligent Cities Business Unit, JD Technology from 2018 to 2021.

Xi Fu is a senior student at the School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University. Her research interests includes Trajectory Data Mining and Machine Learning.



Cheng Long (S'11-M'15) is currently an Assistant Professor at the School of Computer Science and Engineering, Nanyang Technological University. From 2016 to 2018, he worked as a lecturer at Queen's University Belfast, UK. He received his PhD degree from the Hong Kong University of Science and Technology, Hong Kong, in 2015, and his BEng degree from South China University of Technology, China, in 2010. His research interests are broadly in data management, data mining and big data analytics.

Zi Xiong received his B.E. degree from Wuhan University in 2020, and is now a M.S. student in Wuhan University. His research focuses on Trajectory Data Mining, Information System, and Information Retrieval. He is currently a researcher in Information Retrieval and Knowledge Mining Laboratory, Wuhan University.

Jie Bao got his Ph.D degree in Computer Science from University of Minnesota at Twin Cities in 2014. He worked as a researcher in Urban Computing Group at Microsoft Research Asia from 2014 to 2017. He currently leads the data management department at JD Intelligent Cities Business Unit, JD Technology. His research interests include: Spatio-temporal Data Management/Mining, Urban Computing, and Location-based Services.

Ruiyuan Li received his Ph.D. degree from Xidian University in 2020, B.E. degree and M.S. degree from Wuhan University in 2013 and 2016, respectively. His research focuses on Urban Computing, Spatio-temporal Data Management on the Cloud, and Distributed Computing. He currently leads the spatio-temporal data group at JD Intelligent Cities Business Unit, JD Technology.

Yiheng Chen serves as the senior product manager at JD Logistics, leading the development of Value Supply Chain System. His work focuses on forecasting, planning, data mining and intelligent decision in the field of supply chain and logistics. He received his master's degree from Southwest Jiaotong University in 2016.

Shengnan "Shane" Wu has 15+ years track record of applying big data analytics and algorithms to help businesses optimize performance. He currently serves as the Chief Data & Analytics Officer at JD (Jing Dong) Logistics, leading the development of its overall data infrastructure, products and intelligent decision systems. Previously, he held various positions in both private and public sectors globally. Dr. Wu is the inventor of several U.S. and China patents, published articles in prestigious journals, and delivered speeches in professional conferences world-wide. He received his Ph.D. degree in Operations Research from the University of Pittsburgh and a B.Eng. from Tsinghua University, respectively.

Yu Zheng is a Vice President of JD.com and Chief Data Scientist at JD Technology. He also leads the Intelligent Cities Business Unit as the president and serves as the managing director of JD Intelligent Cities Research. His research interests include big data analytics, spatio-temporal data mining, machine learning, and artificial intelligence. Zheng is an ACM Distinguished Scientist and an IEEE Fellow. Before joining JD Technology, he was a senior research manager at Microsoft Research. Zheng is also a Chair Professor at Shanghai Jiao Tong University, an Adjunct Professor at Hong Kong University of Science and Technology.