Computing and Informatics, Vol. 36, 2017, 283–306, doi: 10.4149/cai_2017_2_283

SMOT+NCS: ALGORITHM FOR DETECTING NON-CONTINUOUS STOPS

Francisco Javier Moreno Arboleda, Holver Patiño

Department of Computer and Decision Sciences
Universidad Nacional de Colombia sede Medellín
Carrera 80 No 65-223
Medellín, Colombia
e-mail: {fjmoreno, hepatinoc}@unal.edu.co

Vania Bogorny

Departamento de Informatica e Estatistica
Universidade Federal de Santa Catarina
SC,88040-900, Brazil
Porto Alegre, Brazil
e-mail: vania.bogorny@ufsc.br

Abstract. Several algorithms have been proposed in the last years for discovering stops in trajectories of moving objects. Some methods consider as stops the subtrajectories that i) have speed lower than the average trajectory speed, ii) present significant direction changes, iii) have gaps, or iv) intersect a given spatial region. In these approaches a time constraint should be met for the subtrajectory to be considered as a stop, and this constraint is absolute (it is met or not). Indeed, these approaches consider stops as a continuous subtrajectory. In this paper, we show that for several application domains the stops do not need to be continuous, and the time constraint should be relaxed. In summary, we present the definitions of non-continuous stops and present an algorithm to discover a new kind of stops. We evaluate the proposed algorithm with a running example and real trajectory data, comparing it to the most similar approach in the literature, the SMoT algorithm.

Keywords: Trajectory stops, moving object trajectories, non-continuous stops, semantic trajectories, stops and moves

Mathematics Subject Classification 2010: 68-04, 68P20

1 INTRODUCTION AND MOTIVATION

In recent years, thanks to technologies such as Geographic Information Systems (GIS) [1] and the price reduction of mobile devices, there has been a large increase in the collection of data about the movement of objects such as people, animals, and vehicles. For instance, the number of persons who own mobile phones that incorporate geolocation features is increasing day by day. There are also geolocation systems that indicate the position of a pet [2] as well as many other projects where animals are equipped with geolocation devices, what allows analysts to monitor their nocturnal and migratory behavior [3, 4]. Regarding vehicles, we can find applications for taxis (`https://hailocab.com/nyc`, `http://www.easytaxi.com`) and private vehicles (`https://www.uber.com`), which allow users to track their movement and request their services. With a similar purpose, there exist applications for trucking companies of land cargo [5] and for container tracking [6]. This has contributed to the generation of very large moving object databases [7].

From the data of movement of an object we can reconstruct its trajectory [8]. According to Spaccapietra [9], a trajectory is the record of the evolution of the position of an object that moves in a space for a period of time in order to achieve a goal. For example, a person moving in the afternoon in a shopping mall in order to buy products, an animal that moves at night through the jungle in search of food, and a vehicle that moves during a day in a city to transport merchandise.

The analysis of trajectories is an active research area that may help to understand the behavior of moving objects, both individually and collectively. Laube in [10] presented a set of possible behaviors for moving objects, and a recent survey has shown a summary of the most studied behaviors in trajectory data [11]. This analysis is useful in areas such as tourism, marketing, animal and human migrations, ecology, vehicle route planning, and traffic management, among others.

A trajectory is generally represented by a sequence of observations (x, y, t), where x and y represent the position of the moving object in a space, and t represents the time when the observation was generated. We call this representation *raw trajectory.* The observations are sorted in ascending order according to their time values.

Several works have proposed different representations for trajectories from a more semantic point of view. The most well-known is the representation as a set of *stops* and *moves*, introduced by Spaccapietra et al. [9]. Generally speaking, in this representation stops are the interesting parts of the trajectory and the moves are the trajectory observations between stops. Different methods have been proposed to compute stops and moves from trajectories, focusing on speed [12] and direction [13]. In this paper, we are especially interested in the method SMoT (Stops and Moves of Trajectories) proposed by Alvares in 2007 [14], which computes stops based on *continuous* intersection of a trajectory with interesting geographic data.

We believe that the first step towards discovering interesting patterns from mobility data is the integration of trajectories with geographic information, as proposed by Alvares [14], since geographic data are largely available (e.g. OpenStreetMap, Geo-wiki, Google Maps, among others). We call this geographic data as regions of interest (ROI) or interesting sites.

The SMoT algorithm determines as stops the sites or ROIs where the moving object continuously remained, i.e., spatially intersecting, for more than a minimal amount of time (called threshold Δst). If an observation intersects an interesting site s, the following observations are then analyzed to obtain the total time during which the object remained *continuously* in s. If this time is greater than the threshold Δst, then it is considered that the object had a stop in s, i.e., for purposes of the application it is considered that the object *actually* visited s (it had a stop in s).

The main problem of the SMoT algorithm is that it finds stops only when a sequence of observations *continuously* intersects an interesting site for a minimal amount of time. Indeed, this minimum time constraint is absolute, where it is either met or not. However, let us consider the following example where the threshold Δst for a person to stop at a cinema is set as 90 minutes. Let us suppose that the person stays 60 minutes inside the cinema and leaves for the bathroom for 5 minutes, coming back for more 40 minutes. The SMoT algorithm would not detect the cinema as a stop, although the person stayed there for a total of 100 minutes. We claim that a stop for the cinema should be recorded, even if the trajectory left it for a very short time. In the following subsections we present two real applications to show that discovering non-continuous stops is necessary to solve different problems in different application scenarios.

1.1 Motivating Example: Urban Patrol

Consider a sector w in an urban area with high crime rates. Police must watch this sector to protect the citizens. Police provides the service of urban patrol every day between 8 pm and 6 am using a vehicular unit. This unit also attends emergencies in neighboring sectors, and during these periods the sector w becomes vulnerable.

To ensure the security of sector w, the vehicular unit should remain at least 8 hours per day inside the sector (let us call this time as Δst), although it ideally should remain there the established 10 hours. When the unit goes to cover an emergency in a neighbor sector z, it is expected that the unit remains *outside* sector w, during this exit, at maximum half an hour, in order to minimize the vulnerability of sector w. Let us call this allowed 30 minutes exit as Δat.

That is, if the unit attends an emergency in sector z and returns to sector w in less than 30 minutes, this is equivalent, in practical terms, that the unit did not leave sector w unprotected.

In order to analyze the historical behavior of the unit, from a set of its daily trajectories, it is of interest to determine how many subtrajectories remained inside sector w at least 8 hours (allowing *eventual* exits of less than 30 minutes).

Figure 1 shows an example of the trajectory of a vehicular unit during the 10 hours of a day of service (starting at 7:50 pm and finishing at 6 am). For simplicity of the example, the time between observations (sampling rate) is regular, 10 minutes, but it can be variable in real data. Notice in Figure 1 that the unit left the sector

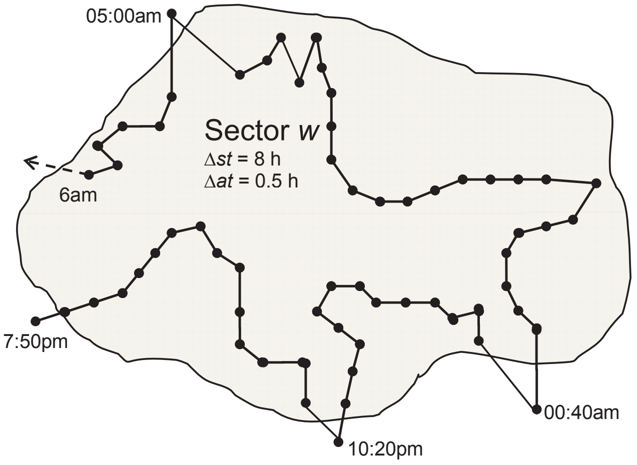

Figure 1. Trajectory of a police vehicular unit in an urban sector w with a non-continuous stop

three times, as explained in Table 1. Table 1 indicates that the unit remained inside sector w (allowing eventual exits less than 30 minutes) for a total of 10 hours, as can be seen in the last column (accumulated time inside sector w). The unit went out of sector w three times, as shown in the central column of Table 1 (at 10:10 pm, 00:30 am, and 04:50 am), but never for more than 30 minutes in each of these three exits. Thus, in practical terms, the unit never left sector w unprotected, and a non-continuous stop of 10 hours is identified.

Continuous interval *inside* sector w	**Continuous interval *outside* sector w**	**Accumulated time *inside* sector w (including the time of the exits)**
[8 pm, 10:10 pm]	(10:10 pm, 10:30 pm)	2 h and 30 min
[10:30 pm, 00:30 am]	(00:30 am, 00:50 am)	4 h and 50 min
[00:50 am, 04:50 am]	(04:50 am, 05:10 am)	9 h and 10 min
[05:10 am, 6 am]	(6 am,. . .)	10 h

Table 1. Movement behavior of the vehicular unit *inside* and *outside* sector w, according to Figure 1

The existing algorithms for computing stops such as CB-SMoT [12], DB-SMoT [13], or SMoT [14] may not discover this stop. The former two methods are respectively based on speed and direction variation, which is not relevant to this problem. On the other hand, for the SMoT algorithm, the minimal amount of time to detect the stop should be eight *continuous* hours or more, so no stops would be recorded in this case. By reducing the amount of time of Δst to one hour, for instance, the SMoT would record four stops and several moves, but the information about how long the object remained inside sector w (including the time of the eventual exits) and for how long the sector was abandoned, would require additional computations and an effort by the analyst adjusting Δst and searching for consecutive series of stops and moves in order to identify non-continuous stops.

In the following section we present a real application scenario where a different type of non-continuous stop is needed to solve the application problem.

1.2 Motivating Example: Congestion Fines

The traffic authorities in London control the number of vehicles that enter downtown between 7 am and 6 pm by a system of congestion fines. Cameras are used to help to identify both entries and exits of the vehicles [15, 16]. Maximum time of continuous stay in the downtown for vehicles is established (Δst). If a vehicle exceeds this threshold, it is charged with a fixed fine (e.g. £ 20) plus a recharge (£ 1 for each minute that it exceeded Δst). The fine is generated when the vehicle leaves downtown and has exceeded the threshold Δst.

In this scenario, a driver could avoid the fine as follows: suppose that $\Delta st = 30$ minutes and a taxi that entered downtown comes out *before* the 30 minutes are met, and after one minute of being outside, it reenters downtown. To avoid this behavior, a second time threshold Δat could be set, to establish that if a vehicle leaves downtown and returns before the threshold Δat is met, it is considered that the vehicle *did not leave* downtown and its staying time inside continues to accumulate. In practical terms this means that the vehicle continues to cause congestion. In this way, drivers are forced to stay out of downtown at least Δat before reentering.

As an example consider Figure 2 and Table 2. As in the previous example, for the sake of simplicity, we consider that the time interval between the points of the trajectory in the Figure is regular (one minute), but for real trajectories the time interval between the points can be variable. Let t be a trajectory with sampling rate one minute, Δst be ten minutes, and Δat be five minutes, and suppose that a vehicle entered downtown at 2 pm, went out at 2:05 pm for one minute and came back at 2:07 pm. A first fine is generated when the vehicle leaves at 2:15 pm (a first non-continuous stop drawn in blue in Figure 2, from 2 pm to 2:15 pm). The amount of the fine is £ 20 + £ 5 (from 2 pm to 2:10 pm was allowed, and from 2:11 to 2:15 exceeded Δst). Note that the exit during one minute is irrelevant in practical terms, i.e., it is as if the vehicle would have been continuously inside downtown from 2 pm to 2:15 pm. For the SMoT algorithm, this one point outside (at 2:06 pm) downtown is enough for not identifying this stop. Next, the vehicle reentered downtown at 2:17 pm (now the staying time of the vehicle is *restarted at zero* because a fine was already generated at 2:16 pm), went out at 2:19 pm for one minute, reentered at 2:21 pm, went out again at 2:23 pm for two minutes, reentered at 2:26 pm, and finally left downtown at 2:33 pm when a second fine is generated (a second non-continuous stop drawn in red in Figure 2). The amount of this second fine is £ 20 + £ 6 (allowed time from 2:17 pm to 2:27 pm, and exceeding time from 2:28 pm to 2:33 pm).

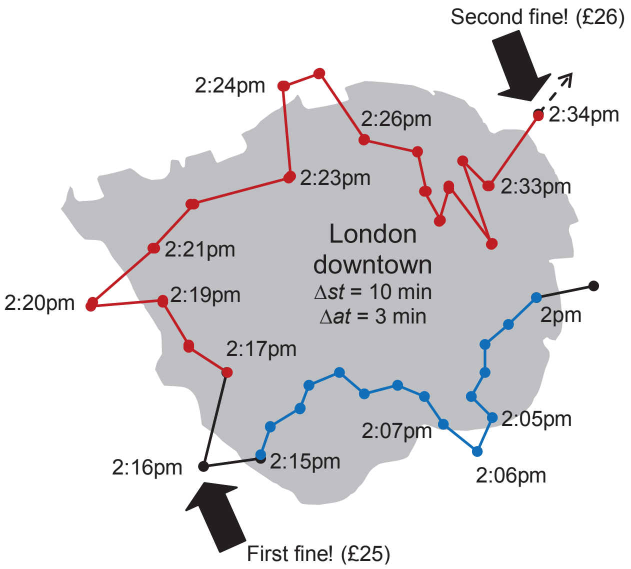

Figure 2. Example of congestion fines: two non-continuous stops (one in blue and the other one in red) generates two corresponding fines

Note that although eventual exits are allowed in both motivating examples and which do not affect the continuity of the stop in the site, as long as the threshold Δat is not exceeded, there is a difference between the two examples. In the first scenario, the stop should be generated *only when the object has exceeded the threshold* Δat at some of its exits of the sector, i.e., the staying time of the object inside the site *continues to accumulate* while the object does not exceed the threshold Δat (note that when the third exit occurs in Figure 1, Δst is already met, but the stop *is not* generated yet). We call this kind of stop as *maximal non-continuous stop.* In the second application, the stop must be generated *right after the object leaves the site and has exceeded the threshold* Δst (this occurs twice in Figure 2 at 2:15 pm and at 2:33 pm and two corresponding fines are generated, the first for £ 25 and the second for £ 26), even if the threshold Δat is not exceeded. We call this kind of stop as *minimal non-continuous stop.*

1.3 Objective and Outline

Having introduced the problem statement through different application examples, in this paper we introduce a new concept and definition of stops in trajectories of moving objects, the non-continuous stops. To compute this new type of stops

Continuous interval *inside* downtown	**Continuous interval *outside* downtown**	**Accumulated time *inside* downtown (including the time of the exits)**	**Fine (Δst exceeded)?**
[2 pm, 2:05 pm]	(2:05 pm, 2:07 pm)	7 min	No
[2:07 pm, 2:15 pm]	(2:15 pm, 2:17 pm)	17 min	Yes → Non-continuous stop: [2 pm, 2:15 pm] Fine: £ 25
[2:17 pm, 2:19 pm]	(2:19 pm, 2:21 pm)	4 min	No
[2:21 pm, 2:23 pm]	(2:23 pm, 2:26 pm)	8 min	No
[2:26 pm, 2:33 pm]	(2:33 pm,. . .)	16 min	Yes → Non-continuous stop: [2:17 pm, 2:33 pm] Fine: £ 26
			Total in fines: £ 51

Table 2. Movement behavior of a vehicle *inside* and *outside* downtown London, according to Figure 2

we propose the algorithm SMoT + NCS (Stops and Moves of Trajectories for Non-Continuous Stops), which can identify the same stops of the method SMoT and two different types of non-continuous stops, *maximal* and *minimal non-continuous stops, i.e. stops with different semantics, and allows relaxing the fulfillment* of Δst using a second threshold Δat.

The rest of the paper is organized as follows. Section 2 presents the related works. Section 3 presents the main definitions and the algorithm. Section 4 presents the experimental evaluation, showing a running example to compare the SMoT and the SMoT + NCS algorithms and an experiment over real trajectories. Finally, Section 5 presents the conclusion and suggests directions of future research.

2 RELATED WORKS

There are several works in the literature for discovering popular places [17, 18], hotspots [19, 20], or regions of interest [21], but these concepts are different from stops. While works on trajectory stops evaluate individual trajectories, hotspots and popular places are computed based on several trajectories. In [17] the method detects regular visits according to the number of times an object visits a place (e.g. an elephant who visits a lake weekly). In [18], a region is defined as popular when a minimal number of trajectories intersects a computed area. In [19], for instance, no a priori information about sites is given, and a hotspot is defined as a place with multiple visits, similarly to the works for finding popular places.

A hotspot should have a minimum size and the trajectory inside should have a minimal length. In [20] a hotspot is identified when a minimal number of trajectories visited a pre-defined region of interest (ROI). In [21], a region is defined as a region of interest when a minimal number of trajectories are close in space (dense area) for a minimal amount of time and with low speed. All previous approaches consider multiple trajectories in their analysis for defining a hotspot, a popular place or a ROI, and these trajectories can be either of the same object or different ones, while the works that specifically focus on stops analyze individual trajectories.

There are only a few works in trajectory literature that define and look for trajectory stops. Spaccapietra in [9] introduced the concept of stops in trajectories, defining the model of *stops* and *moves*, where stops are the important parts of a trajectory from the application point of view and where the object has remained for a minimal amount of time. A more generalized version of this model is presented in [22].

Considering the idea of stops introduced in [9], three main algorithms have been proposed to instantiate the model of stops. Alvares in [14] proposed the algorithm SMoT (which we extend in this paper). The algorithm takes as input a set of user defined regions of interest (e.g. hotels, restaurants) and a set of trajectories. When a trajectory intersects a ROI for a minimal amount of time, the label of the ROI is added to the subtrajectory that continuously intersects the ROI. So it identifies stops in *individual* trajectories. This algorithm is extended in [23] for detecting stops in aggregated ROIs, using concept hierarchies to detect stops at different granularity levels.

A second algorithm for detecting stops is proposed by Palma [12], named CB-SMoT, which determines the stops (they are referred to as clusters) based on the speed of the trajectory. The essential consideration of the CB-SMoT algorithm is that the parts of a single trajectory in which the average speed is lower than a user specified threshold, correspond to the stops (e.g. a tourist that travels through a city and stops at noon for one hour in a restaurant). Rocha in [13] proposed the DB-SMoT algorithm, which determines the stops of an object based on the direction variation of the trajectory. These algorithms consider stops as *continuous*, where an object should continuously intersect or remain inside the same ROI for a minimal amount of time. Yan [24] proposed to use ontologies for discovering stops, where the ontology stores information about the geographic space, the application, and the trajectory. These three components allow performing queries over the ontology to discover the stops, but only for continuous stops, as in Alvares [14].

We showed with real application examples that stops cannot always be continuous, and that the absolute minimal time constraint is too rigid to detect stops at places that an object visits several times, but with short exits. To overcome this problem we introduce the definition of non-continuous stops and use a threshold that allows an object to leave a ROI (site) for a short time, and the time of the stops keeps accumulating when the object leaves the ROI and reenters after a small period of time.

In the following section we present the main definitions and the SMoT + NCS algorithm.

3 SMOT + NCS: EXTENDING THE SMOT ALGORITHM FOR NON-CONTINUOUS STOPS

The SMoT + NCS algorithm is an extension of the SMoT algorithm for identifying *non-continuous stops* (NCS). Before going into details of the algorithm, let us start with some definitions.

3.1 Main Concepts

Trajectory: A trajectory T of a moving object is a sequence of consecutive observations in time $\langle Obs_1, Obs_2, \ldots, Obs_n \rangle$, where $Obs_i = (x_i, y_i, t_i), x_i, y_i \in \Re$, $t \in \Re^+$, $t_1 < t_2 < \ldots < t_n$. x_i and y_i represent the position of the object in $\Re^2$ and t_i the time when the observation was generated.

In order to be able to analyze small parts of trajectories we define the concept of subtrajectory.

Subtrajectory: Let $T = \langle Obs_1, Obs_2, \ldots, Obs_n \rangle$ be a trajectory. A sequence $S = \langle Obs_k, Obs_{k+1}, \ldots, Obs_{k+j} \rangle$ is a subtrajectory of T *iff* $t_k \geq t_1, t_{k+j} \leq t_n$ and $\forall$ $Obs_i \in T$, $t_i \geq t_k \wedge t_i \leq t_{k+j} \rightarrow Obs_i \in S$.

Interesting site: An interesting site s is a tuple $(R_s, \Delta st_s, \Delta at_s)$ where R_s represents the geometry of the site (a topologically closed polygon), Δst_s is the minimum accumulated time to consider that the site has *actually* been visited, and Δat_s is the minimum time to consider that the site was *actually* abandoned by the object. In other words, if an object leaves the site s and returns without having exceeded the threshold Δat_s then, it is considered that the object did not abandon the site.

In Table 3 we show possible values for the thresholds Δst and Δat for different scenarios. In order to make our approach flexible for any application domain we define *application*.

Application: An application $\mathcal{A}$ is a set of non-overlapping interesting sites, such that $\mathcal{A} = \{s_1, s_2, \ldots, s_n\}$.

Continuous stop and substop: Let T be a trajectory of an object and let $\mathcal{A} = \{(R_1, \Delta st_1, \Delta at_1), (R_2, \Delta st_2, \Delta at_2), \ldots, (R_r, \Delta st_r, \Delta at_r)\}$ be an application. Let Tp be the maximum subtrajectory $\langle (x_m, y_m, t_m), (x_{m+1}, y_{m+1}, t_{m+1}), \ldots, (x_{m+k}, y_{m+k}, t_{m+k}) \rangle$ of T, such that exists a $(R_i, \Delta st_i, \Delta at_i) \in \mathcal{A}$ such that $\forall j \in [m, m+k]$ the point (x_j, y_j) is inside the polygon R_i. Then:

- If $|t_{m+k} - t_m| \geq \Delta st_i$, Tp is a *continuous* stop of the object in i.
- If $|t_{m+k} - t_m| < \Delta st_i$, Tp is a *substop* of the object in i.

Scenario	**Δst**	**Δat**	**Observations**
Cinema	90 min	5 min	$\Delta st \rightarrow$ It represents the average duration time of a movie. $\Delta at \rightarrow$ For example, a person goes to the bathroom or answers a phone call (outside the cinema). If the person exceeds this threshold, it could be considered that he/she did not watch completely the movie.
Supermarket	60 min	5 min	$\Delta st \rightarrow$ It represents the average time that a person needs to make his/her purchases. $\Delta at \rightarrow$ For example, a person goes to an ATM (located outside the supermarket) to withdraw money or to the bathroom. If the person exceeds this threshold, it could be considered that he/she left the supermarket.
Sector of a city, e.g., the downtown	30 min	5 min	$\Delta st \rightarrow$ It represents the maximum time that a vehicle may remain in the sector without generating congestion. $\Delta at \rightarrow$ If a vehicle leaves the sector and exceeds this threshold, it could be considered that actually it is out of the sector.
Jungle	8 h	2 h	$\Delta st \rightarrow$ It represents the time that an animal must remain within a region to mark its territory. $\Delta at \rightarrow$ If the animal leaves the region (e.g. for searching food) and exceeds this threshold, the risk that other animals try to seize its territory is high, because it is considered that the animal abandoned the region.

Table 3. Examples of values for Δat and Δst for different application scenarios

Having defined continuous stops and substops we are able to define non-continuous stops. To avoid redundancy, we directly define two different types of *non-continuous stops*, to solve the two different problems introduced at the beginning of the paper: *minimal* and *maximal non-continuous stops.*

Minimal non-continuous stop: Let T be a trajectory of an object and let $\mathcal{A} = \{(R_1, \Delta st_1, \Delta at_1), (R_2, \Delta st_2, \Delta at_2), \ldots, (R_r, \Delta st_r, \Delta at_r)\}$ be an application. Let *endTime* and *startTime* be functions that return the least and the greatest time t of a subtrajectory, respectively. *Let* $\langle \mathit{Subtraj}_1, \mathit{Subtraj}_2, \ldots, \mathit{Subtraj}_n \rangle$, $n \geq 2$ be the **minimal sequence of subtrajectories** on a site $i \in \mathcal{A}$ such that:

a) $\mathrm{endTime}(\mathit{Subtraj}_n) - \mathrm{startTime}(\mathit{Subtraj}_1) > \Delta st_i$.
b) $\mathrm{startTime}(\mathit{Subtraj}_j) - \mathrm{endTime}(\mathit{Subtraj}_{j-1}) \leq \Delta at_i$, $\forall j = 2, \ldots, n$.
c) $\mathit{Subtraj}_j$, $\forall j = 1, \ldots, n-1$, is a substop.
d) $\mathit{Subtraj}_n$ is a substop or a continuous stop.

The subtrajectory of T starting at point startTime($\mathit{Subtraj}_1$) and ending at endTime($\mathit{Subtraj}_n$) is a *minimal non-continuous stop.*

Condition a) specifies that the total temporal extension of the sequence of subtrajectories must exceed Δst_i and condition b) states that for each pair of consecutive subtrajectories $j-1$ and j, the time that elapsed between the end time of the subtrajectory $j-1$ and the start time of the subtrajectory j *must not* exceed the threshold Δat_1

Maximal non-continuous stop: Let T be a trajectory of an object and let $\mathcal{A} = \{(R_1, \Delta st_1, \Delta at_1), (R_2, \Delta st_2, \Delta at_2), \ldots, (R_r, \Delta st_r, \Delta at_r)\}$ be an application. Let *endTime* and *startTime* be functions that return the least and the greatest time t of a subtrajectory, respectively. Let $\langle \mathit{Subtraj}_1, \mathit{Subtraj}_2, \ldots, \mathit{Subtraj}_n \rangle$, $n \geq 2$ be the **maximal sequence of subtrajectories** on a site $i \in \mathcal{A}$ such that:

a) endTime($\mathit{Subtraj}_n$) − startTime($\mathit{Subtraj}_1$) $> \Delta st_i$.
b) startTime($\mathit{Subtraj}_j$) − endTime($\mathit{Subtraj}_{j-1}$) $\leq \Delta at_i$, $\forall j = 2, \ldots, n$.
c) $\mathit{Subtraj}_j$,$\forall j = 1, \ldots, n$, is a substop or a continuous stop.

The subtrajectory of T starting at point startTime($\mathit{Subtraj}_1$) and ending at endTime($\mathit{Subtraj}_n$) is a *maximal non-continuous stop.*

Figure 3 illustrates these definitions, where we show both minimal and maximal NCS, considering a trajectory which has four substops (from SubS_1 to SubS_4). According to Figure 3, a *minimal* NCS includes the sequence of substops $\langle \mathit{SubS}_1, \mathit{SubS}_2, \mathit{SubS}_3 \rangle$ generated when Δst is exceeded and Δat is not. A *maximal* NCS includes the sequence of substops $\langle \mathit{SubS}_1, \mathit{SubS}_2, \mathit{SubS}_3, \mathit{SubS}_4 \rangle$, where the time of the NCS keeps accumulating even when Δst is met, until Δat is exceeded.

Having defined the stops, the parts of trajectories that are not stops (where the stops can be continuous or not) are moves, as defined as follows.

Move: A move of T with regard to $\mathcal{A}$ is one of the following cases: i) the maximum subtrajectory between two consecutive stops, ii) the maximum subtrajectory between the first observation of T and the first stop, iii) the maximum subtrajectory between the last stop and the last observation of the trajectory, iv) the same trajectory T, if T has no stops.

In the following section we present the proposed algorithm.

3.2 The SMoT + NCS Algorithm

The SMoT + NCS algorithm, presented in Listing 1, identifies both *continuous* and *non-continuous* stops from a set of trajectories T and an application $\mathcal{A}$. The algorithm SMoT+NCS receives a third parameter called *type*, whose value is *Minimal* for considering minimal non-continuous stops and *Maximal* for maximal non-continuous stops. This parameter is application dependent, such that the user may choose the

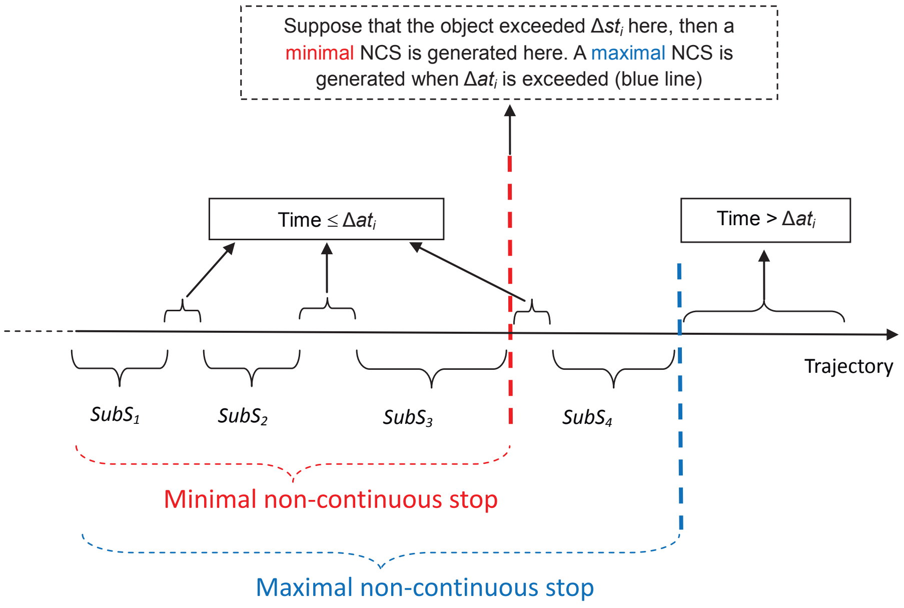
Suppose that the object exceeded Δst_i here, then a minimal NCS is generated here. A maximal NCS is generated when Δat_i is exceeded (blue line)
Time ≤ Δat_i
Time > Δat_i
Trajectory
SubS_1
SubS_2
SubS_3
SubS_4
Minimal non-continuous stop
Maximal non-continuous stop

Figure 3. Examples of minimal and maximal NCS

type of non-continuous stops he wants to extract. In the following we explain how the algorithm works for both types:

- type = Minimal: The SMoT + NCS algorithm generates a stop (continuous or non-continuous) as soon as the object exceeds the threshold Δst of a site i even if Δat is not exceeded;
- type = Maximal: The SMoT + NCS algorithm generates a stop (continuous or non-continuous) only when the object exceeds both thresholds Δst and Δat of a site i.

We present the pseudo-code of SMoT + NCS as an extension of SMoT, where the first step is to find continuous stops such as the method SMoT, and the new steps to find NCS are computed in lines 22 to 25, 31 to 37 and 38 to 42.

Basically, the SMoT + NCS algorithm works as follows: through a loop, the algorithm traverses each of the trajectories T of the set of trajectories T (line 6). Through a second loop it traverses each of the observations of a trajectory T (line 9). It checks if there is an interesting site $(R_s, \Delta st_s, \Delta at_s) \in \mathcal{A}$ such that the current observation (x, y) of the trajectory intersects it (line 10). If so, it continues with a loop (line 15) that traverses each observation of the trajectory that intersects the current interesting site and where the criteria are verified to determine if this is a stop. It checks if the accumulated time in the interesting site exceeds the established time (Δst_s) to define if this is a stop (line 23). It also checks if the

Algorithm 1 SMoT + NCS algorithm

```
1: INPUT: $\mathcal{T}$ : Set of trajectories, $\mathcal{A}$ : Application, type : Type of the NCS
2: OUTPUT: $\mathcal{S}$ : Set of stops, $\mathcal{M}$ : Set of moves
3: METHOD:
4: $\mathcal{S}$ = new Stops();
5: $\mathcal{M}$ = new Moves();
6: for each trajectory T $\in \mathcal{T}$ do
7:    $i = 1$;
8:    prevStop = null;
9:    while ($i \leq$ size(T)) do
10:      if ($\exists(R_s, \Delta st_s, \Delta at_s) \in \mathcal{A}$ | point(T[$i$]) intersects $R_s$) then // Using spatial index
11:         enterTime = time($T[i]$);
12:         breakTime = enterTime;
13:         accumulTime = 0;
14:         currentSite = $(R_s, \Delta st_s, \Delta at_s)$;
15:         while (intersects(T[$i$], currentSite.$R_s$)) do // Object enters $R_s$
16:            while (intersects($T[i]$, currentSite.$R_s$)) do
17:               $i++$; // Object remains inside $R_s$
18:            end while
19:            $i--$; // Go one step back (went outside currentSite.$R_s$)
20:            ileave = $i$;
21:            leaveTime = time($T[i]$);
22:            accumulTime = leaveTime − enterTime;
23:            if (type = "Minimal" AND accumulTime $\geq$ currentSite.$\Delta st_s$) then
24:               break; // Exit while
25:            end if
26:            if ($i$ = size($T$)) then
27:               break; // Exit while
28:            end if
29:            $i++$;
30:            breakTime = time($T[i]$);
31:            while (breakTime − leaveTime $\leq$ currentSite.$\Delta at_s$ AND $i <$ size($T$)) do // Object left $R_s$
32:               $i++$;
33:               breakTime = time($T[i]$);
34:               if (intersects($T[i]$, currentSite.$R_s$)) then // Object came back to $R_s$
35:                  break; // Exit while
36:               end if
37:            end while
38:            if (breakTime − leaveTime > currentSite.$\Delta at_s$
39:              OR $i$ = size($T$)) then
40:              $i$ = ileave;
41:              break; //Exit while
42:            end if
43:         end while
44:         if (accumulTime $\geq$ currentSite.$\Delta st_s$) then
45:            stop = ($T$, currentSite.$R_s$, enterTime, leaveTime);
46:            $\mathcal{S}$.add(stop);
47:            move = ($T$, prevStop, stop, prevStop.leaveTime, enterTime);
48:            $\mathcal{M}$.add(move);
49:            prevStop = stop;
50:         end if
51:      end if
52:      $i++$;
53:   end while
54:   if ($T[i-1]$ not $\in$ prevStop) then
55:      move = ($T$, prevStop, null, prevStop.leaveTime, time($T[i-1]$));
56:      $\mathcal{M}$.add(move);
57:   end if
58: end for
```

current observation is the last one of T (line 26). It enters to another loop (lines 31 to 37) where the observations of the trajectory are traversed while both Δat_s and the size of the trajectory are not exceeded. This loop also checks if the trajectory entered back to the interesting site (line 34). After these verifications, it analyzes if the accumulated time exceeds the threshold Δst_s (line 44) to define if it is a stop, and if so, it is added to the set of stops (line 46).

As for the SMoT algorithm, for the SMoT + NCS we assume that the interesting sites in the application $\mathcal{A}$ are non-overlapping. They can touch each other, as can be seen in the experiments, but should not overlap.

Because most of the runtime is needed for the intersection operation (lines 10, 15, 16, and 34), which must be performed for each point of a trajectory and because our algorithm is restricted to mutually non-overlapping regions, we use an auxiliary array in the implementation to compute the possible intersection of each point with a region. In doing so, the intersection operation is performed just once for each point, using a spatial index. The complexity of the intersection operation in the algorithm is $O(m * n)$, where m is the number of trajectories (cardinality of $\mathcal{T}$) and n denotes the length of a trajectory.

The best and average-case complexity of the SMoT + NCS algorithm is $O(m * n)$. The theoretical worst-case complexity is $O(m * n^2)$ and it may occur when detecting maximal NCS and the algorithm must re-traverse *almost all* the observations of a trajectory, but in practice, this case is unusual. It is important to point out that for continuous stops the algorithm does not have to re-traverse any observation. In addition, most of the runtime is for the spatial intersection, whose complexity is $O(m * n)$.

4 EXPERIMENTAL EVALUATION

In this section, we present some experiments to show the efficacy of our method and compare the results with the SMoT method. In Section 4.1 we show an example of stops detected by both SMoT and SMoT + NCS algorithms considering four interesting sites. In Section 4.2 we present the results of the experiments performed over a real trajectory dataset collected in the city of Rio de Janeiro.

4.1 A Running Example to Compare the SMoT and the SMoT + NCS Algorithms

Consider the application $\mathcal{A} = \{(R_{s1}, 40\,\text{min}, 10\,\text{min}), (R_{s2}, 30\,\text{min}, 20\,\text{min}), (R_{s3}, 30\,\text{min}, 20\,\text{min}), (R_{s4}, 15\,\text{min}, 15\,\text{min})\}$ with four interesting sites (s_1, s_2, s_3, and s_4) and the trajectory of Figure 4 (starting at observation 1 and finishing at observation 79). The time between observations is five minutes. After executing the SMoT and the SMoT + NCS algorithms we obtained the results of Table 4.

Let us start analyzing the site s_1 in Figure 4 to understand how the algorithm proceeds. The SMoT + NCS algorithm begins its analysis at observation 1. As this does not intersect the polygon of any interesting site, it goes on to the observation 2 that intersects R_{s1}. Then, the algorithm records the time of entry at observation 2, the data of the interesting site (s_1), and begins to accumulate time until observation 7. It checks if the accumulated time is equal or exceeds the threshold Δst_{s1} (40 minutes). As this time is not exceeded (it is 25 minutes), the algorithm continues with the observation 8 and checks that the time since leaving s_1 (observation 8) does not exceed the threshold Δat_{s1} (10 minutes). In observation 9 it checks if the trajectory reentered to s_1 and continues to accumulate time in s_1 until observation 12, when Δst_{s1} is reached. For both minimal and maximal NCS, a stop is generated from observations 2–12 and a move from observations 1–2 and 12–15 (see Table 4 for more details). So in this site, no stop is recorded by the algorithm SMoT, only a long move.

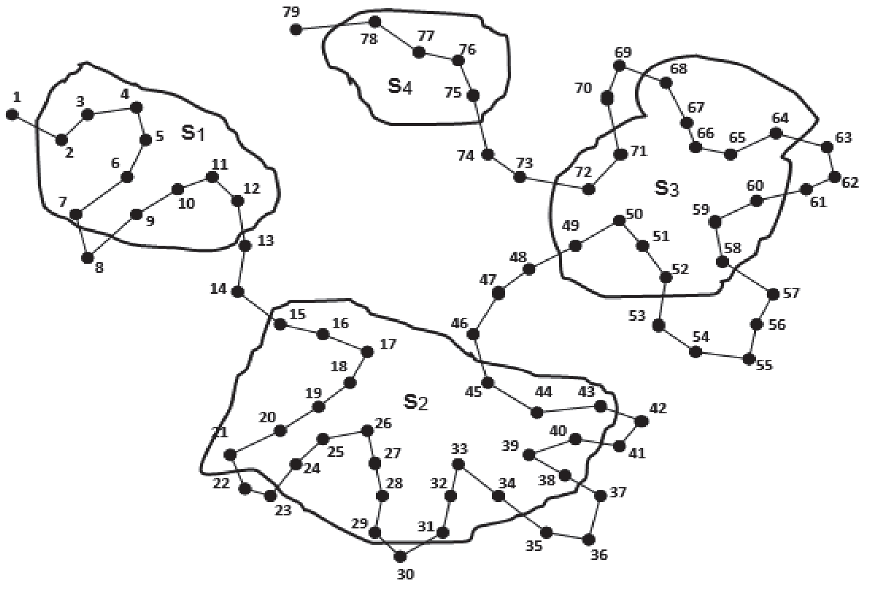

Figure 4. Trajectory for the running example

The site s_2 of Figure 4 is more interesting, because it clearly shows the difference between SMoT and SMoT + NCS. As can be seen in Table 4, SMoT generates one stop for s_2 from observations 15–21, where $\Delta st_{s2} = 30$ minutes is reached. For the remaining observations in this site it generates a move. Our algorithm for *minimal* NCS will detect a stop as soon as Δst_{s2} is reached, so generating a stop from observations 15–21 (when $\Delta st_{s2} = 30$ minutes is reached the first time) and from observations 24–34, when Δst_{s2} is again reached without exceeding Δat_{s2}. After observation 34 no other stop for $\Delta st_{s2} = 30$ minutes is detected, so generating two stops and two moves (see Table 4). Our algorithm for *maximal* NCS keeps

Observations	**Site**	**SMoT**	**SMoT + NCS**	
			Type = Minimal	**Type = Maximal**
[1–2)		Move	Move	Move
[2–7]	s_1		Stop(s_1)	Stop(s_1)
(7–9)				
[9–12]	s_1			
(12–15)			Move	Move
[15–21]	s_2	Stop(s_2)	Stop(s_2)	Stop(s_2)
(21–24)		Move	Move	
[24–29]	s_2		Stop(s_2)	
(29–31)				
[31–34]	s_2			
(34–38)			Move	
[38–40]	s_2			
(40–43)				
[43–45]	s_2			
(45–49)				Move
[49–52]	s_3			
(52–58)				
[58–60]	s_3		Stop(s_3)	Stop(s_3)
(60–64)				
[64–68]	s_3			
(68–71)			Move	
[71–72]	s_3			
(72–75)				Move
[75–78]	s_4	Stop(s_4)	Stop(s_4)	Stop(s_4)
(78–79]		Move	Move	Move

Table 4. Results for the running example for the SMoT and the SMoT + NCS algorithms

accumulating time from the observations 15–45, even when the object leaves s_2 at observations (22–23, 30, 35–37, and 41–42), because Δat_{s2} is never greater than 20 minutes at each exit. So a maximal NCS is generated from observations 15–45, as can be seen in Table 4.

In summary, for site s_2, the three approaches generate different types of stops and moves, clearly showing the contribution and flexibility of our method.

By considering all four sites in Table 4, we can notice that for both SMoT and SMoT + NCS algorithms the number and size of stops are different. For all sites, considering the respective Δat and Δst of each site, SMoT generated only two stops in total, while SMoT + NCS generated five stops (for type = *minimal*) and four stops (for type = *maximal*). SMoT identifies only two stops, one in s_2 and one in s_4, because only in these sites the trajectory has at least one continuous subtrajectory remaining inside the sites for more than Δst.

It is worth mentioning here that as the proposed algorithm SMoT + NCS is a generalized version of SMoT, it detects three kinds of stops: continuous stops, minimal NCS and maximal NCS). If we consider $\Delta at = 0$, i.e., not allowing any exits, our algorithm will find the same stops as SMoT.

4.2 Experimental Evaluation with Real Trajectories

For traffic managers and urban planners it is important to know the amount of cars that stay inside a certain area in a city, for different purposes, such as to understand people's mobility in the area, to measure air pollution, to estimate parking lots, and so on. Therefore, the experiments were conducted with 100 trajectories of vehicles circulating in Rio de Janeiro (Brazil). We considered 137 districts as the interesting sites, where these districts are touching regions. The total number of observations was 268 000, i.e., an average of 2 680 observations per trajectory. The average time between two consecutive observations is two seconds. Thus, the average duration of each trajectory was one and half an hour.

The experiments were conducted on a computer with Intel Core i3 M370 of 2.40 GHz with DDR2 RAM 8 GB. The algorithms (SMoT and SMoT + NCS) were implemented in the programming language PLPGSQL of PostgreSQL 8.4, that were integrated with PostGIS 8.4 and QGIS 2014 for visualization aspects. In Figure 5 we show a map of Rio de Janeiro with its districts and one of the analyzed trajectories (dark black lines).

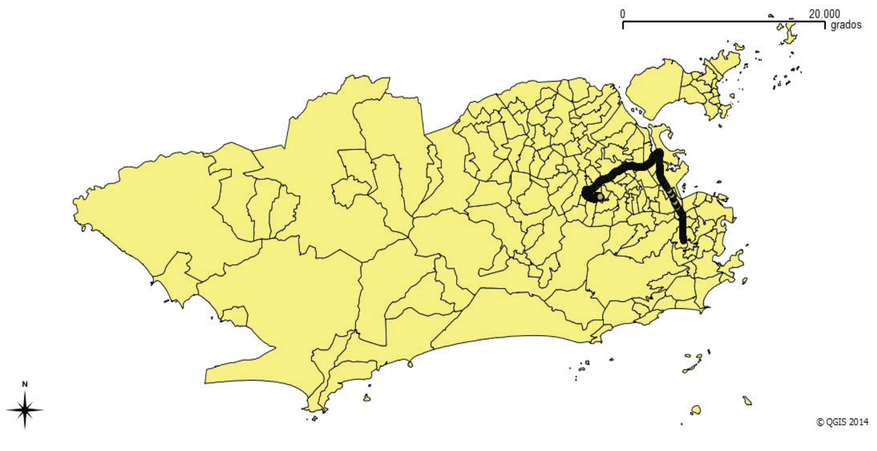

Figure 5. Map of Rio de Janeiro, its neighborhoods, and one of the analyzed trajectories

In Figure 6 we show an enlarged fragment of the map in Figure 5 and example of a stop not detected by SMoT in this dataset. Note that the trajectory enters the district marked with an asterisk, then it goes out of it (skirting it), reenters, and finally traverses through it.

We performed several experiments on this dataset to compare our method with SMoT, considering the same Δst for *all districts*. First, we considered $\Delta st = 2$

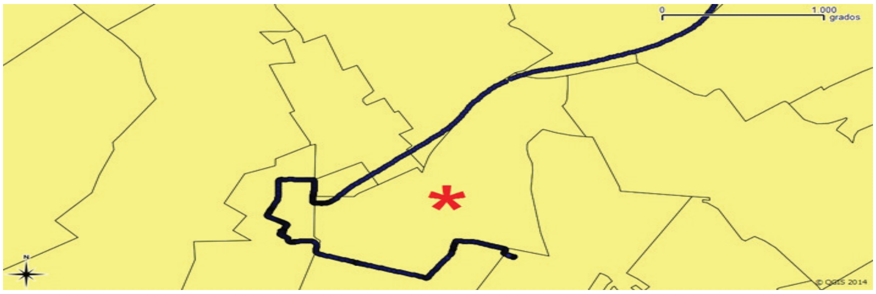

Figure 6. Enlarged fragment of the map from Figure 5

minutes and Δat = 1 minute. The results are shown in Table 5 and will be explained later. Then, we changed Δst to 4 minutes, 8 minutes, and 16 minutes, as shown in Tables 6, 7 and 8, respectively. In all experiments we analyzed the following information, as can be seen in the tables: the total number of stops, the total number of NCS, the average number of stops and NCS of each trajectory, the average time duration of stops and NCS, and finally, the running time of the algorithms.

Indicator	**SMoT**	**SMoT + NCS**	
		Type = Minimal	**Type = Maximal**
Total number of stops	635	702	655
Total number of NCS	–	67	66
Average number of stops by trajectory	6.35	7.02	6.55
Average number of NCS by trajectory	–	0.67	0.66
Average time of stops (min:sec)	05 : 12	05:14	05:24
Average time of NCS (min:sec)	–	06:30	07:15
Execution time (in seconds)	406	514	597

Table 5. Results of the algorithms: SMoT, SMoT + NCS with type = Minimal, and SMoT + NCS with type = Maximal. $\Delta st = 2\,\text{min}$ and $\Delta at = 1\,\text{min}$.

As can be seen in Tables 5 to 8, in the first row, the total number of stops detected by our method SMoT + NCS for type = Minimal (column 3), is always greater or equal to the number of stops detected by SMoT (column 2). This occurs because our method detects all continuous stops detected by SMoT, plus the non-continuous stops, if they exist. On the other hand, for our algorithm with type = Maximal, we cannot anticipate the number of stops in relation to SMoT, because it can find more, for instance, if SMoT does not find any stop, and our method detects a non-continuous stop. Still, for type = Maximal, our method can find the same number of stops as SMoT, as for instance, when SMoT detects a stop, and just after this stop the object leaves the site and Δat is exceeded. Finally, when

SMoT generates two consecutive stops in the same site, and the exit time between these stops does not exceed Δat, then our algorithm will detect one maximal NCS, i.e., less stops than SMoT.

In what concerns to the number of NCS (second row in Table 4), the maximal NCS detected by SMoT + NCS must be equal or less than the number of minimal NCSs. Note that both minimal and maximal NCS must reach the threshold Δst, but a maximal NCS *tends to be longer*, because the time of a NCS continues to accumulate as long as the threshold Δat is not exceeded, while a minimal NCS is generated as soon as the threshold Δst.

Indicator	**SMoT**	**SMoT + NCS**	
		Type = Minimal	**Type = Maximal**
Total number of stops	280	313	302
Total number of NCS	–	33	33
Average number of stops by trajectory	2.80	3.13	3.02
Average number of NCS by trajectory	–	0.33	0.33
Average time of stops (min:sec)	07:59	08:06	08:10
Average time of NCS (min:sec)	–	09:05	09:43
Execution time (in seconds)	428	586	604

Table 6. Results of the algorithms: SMoT, SMoT + NCS with type = Minimal, and SMoT + NCS with type = Maximal. $\Delta st = 4\,\text{min}$ and $\Delta at = 1\,\text{min}$.

Indicator	**SMoT**	**SMoT + NCS**	
		Type = Minimal	**Type = Maximal**
Total number of stops	125	136	124
Total number of NCS	–	11	11
Average number of stops by trajectory	1.25	1.36	1.24
Average number of NCS by trajectory	–	0.11	0.11
Average time of stops (min:sec)	10:26	10:34	11:01
Average time of NCS (min:sec)	–	12:15	17:01
Execution time (in seconds)	450	607	620

Table 7. Results of the algorithms: SMoT, SMoT + NCS with type = Minimal, and SMoT + NCS with type = Maximal. $\Delta st = 8\,\text{min}$ and $\Delta at = 1\,\text{min}$.

The rows 5 and 6 in the tables show the average time duration of the stops and the non-continuous stops. The average time duration of stops detected by SMoT + NCS for type = Maximal is always equal or greater than the average time duration of stops generated by SMoT and SMoT + NCS for type = Minimal.

Indicator	**SMoT**	**SMoT + NCS**	
		Type = Minimal	**Type = Maximal**
Total number of stops	20	23	19
Total number of NCS	–	3	2
Average number of stops by trajectory	0.20	0.23	0.19
Average number of NCS by trajectory	–	0.03	0.02
Average time of stops (min:sec)	18:57	19:02	19:42
Average time of NCS (min:sec)	–	19:40	26:05
Execution time (in seconds)	482	622	643

Table 8. Results of the algorithms: SMoT, SMoT + NCS with type = Minimal, and SMoT + NCS with type = Maximal. $\Delta st = 16$ min and $\Delta at = 1$ min.

As shown in Tables 5, 6, 7, and 8, the execution time of the SMoT + NCS algorithm for detecting maximal NCS was the greatest of the three, because the algorithm sometimes has to go back (see running example) and to traverse again some observations of a trajectory. This usually happens to a lower extent when detecting minimal NCS, and it does not happen in the SMoT algorithm.

In summary, as expected, by increasing the minimal time for a stop, i.e., Δst, the number of stops decreases (e.g. from 635 of SMoT in Table 5 to 20 in Table 8), because the more time an object should spend in a district, the less trajectories will satisfy this threshold. But it is worth mentioning here that the main contribution of this paper is that our method finds different types of stops in relation to SMoT, independently of the minimal time duration of a stop.

The values of both Δst and Δat are application dependent. The specialist of a domain has the knowledge about how to define these parameters. In the application examples presented at the beginning of our paper, the duration of a NCS for the police vehicular unit to monitor a sector should be eight hours (with exits no longer than 30 minutes) and in the congestion fines application the city administrators have a clear rule on how long an object can stay inside a ROI and how long it should stay outside to be able to come back without getting a fine. Similarly, in applications where someone is interested in monitoring the stops of a worker, Δst should be defined as eight hours for a full time worker and four hours for part time worker. Each application has its requirements for the time duration, and we show with our experiments how the results change when the parameter values increase.

5 CONCLUSIONS AND FUTURE WORKS

In this paper, we showed with real application examples, that stops cannot always be continuous, and that the absolute minimal time constraint is too rigid to detect stops at places that an object visits several times, but with short exits. To overcome this problem we introduced the definition of non-continuous stops, and use a threshold that allows an object to leave a site for a short time. We proposed a new approach which gives a new definition of stops, the non-continuous stops (NCS), and proposed an algorithm named SMoT + NCS for detecting continuous stops, maximal non-continuous stops, and minimal non-continuous stops. For discovering maximal NCS the time of the stops keeps accumulating when the object leaves the site and reenters after a (small) period of time. For the minimal NCS a stop is generated as soon as the minimal time duration threshold (Δst) is met.

Our proposed method is a generalized version of the well known algorithm SMoT, and it can detect, in fact, the same continuous stops detected by SMoT and two different types of NCS. We compared the results of our algorithm with SMoT, considering a running example and a real moving object trajectory dataset, clearly showing that SMoT does not find non-continuous stops.

As future work, we consider the following issues:

1. first, we must improve the performance of SMoT + NCS;
2. second, we plan to develop a specialized query language to allow users to formulate expressive and intuitive queries in order to discover patterns from the stops, such as rank the sites according to the number of stops and their duration;
3. third, identify the objects that in their different trajectories have recurrent stops in certain sites, and identify stops that are shared for two or more trajectories; and
4. fourth, as one of the reviewers pointed out, a third threshold to control the effective time of an object inside a ROI could be added

For example, consider this case: a worker is obliged to stay eight hours at his workplace, but he/she is allowed to leave it for less than ten minutes. It meets the definition of NCS, but the worker can have the ten minute breaks one after the other. Thus, he/she may spend his/her workday having mainly breaks!

We also refer the reader to the recent expository paper by Ilarri et al. [25] where a detailed and comprehensive survey in the domain of management of semantic locations and trajectories is given. The authors also outlined research challenges such as semantic representation of moving objects, semantic queries, analysis and mining of semantic mobility data, privacy issues, among others.

REFERENCES

[1] Longley, P. A.—Goodchild, M.—Maguire, D. J.—Rhind, D. W.: Geographic Information Systems and Science. 3rd ed. Wiley, 2010.

[2] Tractive Pet Tracking. Available at: `http://tractive.com/es`.

[3] CORDIS: A Community Information Service for Research and Development: GPS Tracking of Endangered Asian Elephants (in Spanish). Available at: `http://cordis.europa.eu/news/rcn/34659_es.html`. Retrieved on August 26, 2014.

[4] Riding, T. A.—Dennis, T. E.—Stewart, C. L.—Walker, M. M.—Montgomery, J. C.: Tracking Fish Using 'Buoy-Based' GPS Telemetry. Marine Ecology Progress Series, Vol. 377, 2009, pp. 255–262. Available at: `http://www.int-res.com/articles/meps2009/377/m377p255.pdf`, doi: 10.3354/meps07809.

[5] Gadri, R. C.—Chavan, A.—Sonawane, R.—Kamble, S.: Land Vehicle Tracking Application on Android Platform. International Journal of Engineering Research and Applications, Vol. 2, 2012, No. 3, pp. 1978–1982. Available at: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.416.2898&rep=rep1&type=pdf`.

[6] Ahn, S. B.: Container Tracking and Tracing System to Enhance Global Visibility. Proceedings of the Eastern Asia Society for Transportation Studies, Vol. 5, 2005, pp. 1719–1727. Available at: `http://www.easts.info/on-line/proceedings_05/1719.pdf`.

[7] Güting, R. H.—Schneider, M.: Moving Objects Databases. Morgan Kaufmann, 2005.

[8] Laube, P.—Purves, R. S.: How Fast Is a Cow? Cross-Scale Analysis of Movement Data. Transactions in GIS, Vol. 15, 2011, No. 3, pp. 401–418, doi: 10.1111/j.1467-9671.2011.01256.x.

[9] Spaccapietra, S.—Parent, C.—Damiani, M. L.—Fernandes de Macedo, J. A.—Porto, F.—Vangenot, C.: A Conceptual View on Trajectories. Data and Knowledge Engineering, Vol. 65, 2008, No. 1, pp. 126–146, doi: 10.1016/j.datak.2007.10.008.

[10] Laube, P.—Imfeld, S.—Weibel, R.: Discovering Relative Motion Patterns in Groups of Moving Point Objects. International Journal of Geographical Information Science, Vol. 19, 2005, No. 6, pp. 639–668.

[11] Parent, C.—Spaccapietra, S.—Renso, C.—Andrienko, G.—Andrienko, N.—Bogorny, V.—Damiani, M. L.—Gkoulalas-Divanis, A.—Macedo, J. A. F.—Pelekis, N.—Theodoridis, Y.—Yan, Z.: Semantic Trajectories Modeling and Analysis. ACM Computing Surveys, Vol. 45, 2013, No. 4, Article No. 42, doi: 10.1145/2501654.2501656.

[12] Palma, A. T.—Bogorny, V.—Kuijpers, B.—Alvares, L. O.: A Clustering-Based Approach for Discovering Interesting Places in Trajectories. Proceedings of the 2008 ACM Symposium on Applied Computing (SAC '08), New York, 2008, pp. 863–868, doi: 10.1145/1363686.1363886.

[13] Rocha, J. A. M. R.—Times, V. C.—Oliveira, G.—Alvares, L. O.—Bogorny, V.: DB-SMoT: A Direction-Based Spatio-Temporal Clustering Method. 5th IEEE International Conference of Intelligent Systems (IS '10), London, 2010, pp. 114–119.

[14] Alvares, L. O.—Bogorny, V.—Kuijpers, B.—Fernandes de Macedo, J. A.—Moelans, B.—Vaisman, A.: A Model for Enriching Trajectories with Semantic Geographical Information. Proceedings of the 15th ACM International Symposium on Geographic Information Systems (GIS '07), New York, 2007, Article No. 22, pp. 1–8.

[15] Leape, J.: The London Congestion Charge. Journal of Economic Perspectives, 2006, pp. 157–176. Available at: `http://pubs.aeaweb.org/doi/pdfplus/10.1257/jep.20.4.157`, doi: 10.1257/jep.20.4.157.

[16] Transport for London. 2013, February 8, retrieved April 26, 2014, from Seeking Your Views about Proposed Changes to the Congestion Charging Scheme. Available at: `https://consultations.tfl.gov.uk/roads/congestioncharging`.

[17] Djordjevic, B.—Gudmundsson, J.—Pham, A.—Wolle, T.: Detecting Regular Visit Patterns. Algorithmica, Vol. 60, 2011, pp. 829–852, doi: 10.1007/s00453-009-9376-2.

[18] Benkert, M.—Djordjevic, B.—Gudmundsson, J.—Wolle, T.: Finding Popular Places. International Journal on Computational Geometry Applications, Vol. 20, 2010, No. 1, pp. 19–42, doi: 10.1142/s0218195910003189.

[19] Gudmundsson, J.—van Kreveld, M. J.—Staals, F.: Algorithms for Hotspot Computation on Trajectory Data. Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL '13), New York, 2013, pp. 134–143, doi: 10.1145/2525314.2525359.

[20] Hoteit, S.—Secci, S.—Sobolevsky, S.—Ratti, C.—Pujolle, G.: Estimating Human Trajectories and Hotspots Through Mobile Phone Data. Computer Networks, Vol. 64, 2014, pp. 296–307, doi: 10.1016/j.comnet.2014.02.011.

[21] Uddin, M. R.—Ravishankar, C. V.—Tsotras, V. J.: Finding Regions of Interest from Trajectory Data. 12th IEEE International Conference on Mobile Data Management (MDM '11), Lulea, June 2011, Vol. 1, pp. 39–48.

[22] Bogorny, V.—Renso, C.—Ribeiro de Aquino, A.—de Lucca Siqueira, F.—Alvares, L. O.: CONSTAnT – A Conceptual Data Model for Semantic Trajectories of Moving Objects. Transactions in GIS, Vol. 18, 2014, No. 1, pp. 66–88, doi: 10.1111/tgis.12011.

[23] Moreno, F.—Pineda, A.—Fileto, R.—Bogorny, V.: SMoT+: Extending the SMoT Algorithm for Discovering Stops in Nested Sites. Computing and Informatics, Vol. 33, 2014, No. 2, pp. 327–342.

[24] Yan, Z.—Macedo, J. A. F.—Parent, C.—Spaccapietra, S.: Trajectory Ontologies and Queries. Transactions in GIS, Vol. 12, 2008, pp. 75–91, doi: 10.1111/j.1467-9671.2008.01137.x.

[25] Ilarri, S.—Stojanovic, D.—Ray, C.: Semantic Management of Moving Objects: A Vision Towards Smart Mobility. Expert Systems with Applications, Vol. 42, 2015, pp. 1418–1435, doi: 10.1016/j.eswa.2014.08.057.

Francisco Javier Moreno Arboleda holds his Ph.D. and a M.Sc. in engineering systems from Universidad Nacional de Colombia. He currently works as Associate Professor at Universidad Nacional de Colombia, sede Medellín. His research areas are data warehouses and spatiotemporal databases.

Vania Bogorny is Professor at Departamento de Informatica e Estatistica of Universidade Federal de Santa Catarina (UFSC), Brazil. She received her M.Sc. (2001) and Ph.D. (2006) in computer science from Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil. In 2007, she received the Best Ph.D. Thesis Award from the Brazilian Computer Society. She has published her papers in refereed journals and conference proceedings, such as the International Conference on Data Mining (IEEE ICDM), International Symposium on Advances in Geographic Information Systems (ACMGIS), International Conference on Intelligent Systems (IEEE IS), Geoinformatica, Transactions in GIS and International Journal of Geographical Information Science (IJGIS). She has served as a reviewer of international journals as IJGIS, Transportation Research Part C, Geoinformatica, TKDE, DKE, Transactions in GIS and DMKD. Her general areas of interest are databases, spatial and spatiotemporal data mining, spatial data modeling, and geographic information systems.

Holver Patiño holds his degree in engineering systems from Universidad Nacional de Colombia. He currently works as an independant international consultant in Oracle. His research areas are algorithms, data warehousing, and spatiotemporal databases.