Computing and Informatics, Vol. 33, 2014, 327–342

SMOT+: EXTENDING THE SMOT ALGORITHM FOR DISCOVERING STOPS IN NESTED SITES

Francisco Moreno, Andrés Pineda

Departamento de Ciencias de la Computación y de la Decisión
Universidad Nacional de Colombia, Sede Medellín
e-mail: {fjmoreno, afpinedac}@unal.edu.co

Renato Fileto, Vania Bogorny

Departamento de Informatica e Estatística
of Universidade Federal de Santa Catarina (UFSC) Brazil
e-mail: {r.fileto, vania.bogorny}@ufsc.br

Abstract. Several methods have been proposed to analyse trajectory data. However, a few of these methods consider trajectory relations with relevant features of the geographic space. One of the best-known methods that take into account the geographical regions crossed by a trajectory is the SMoT algorithm. Nevertheless, SMoT considers only disjoint geographic regions that a trajectory may traverse, while many regions of interest are contained in other regions. In this article, we extend the SMoT algorithm for discovering stops in nested regions. The proposed algorithm, called SMoT+, takes advantage of information about the hierarchy of nested regions to efficiently discover the stops in regions at different levels of this hierarchy. Experiments with real data show that SMoT+ detects stops in nested regions, which are not detected by the original SMoT algorithm, with minor growth of processing time.

Keywords: Trajectories of moving objects, stops and moves, semantic trajectories, nested sites, trajectory episodes in different spatial granularities

1 INTRODUCTION

The increasing use of mobile devices, such as GPS and cell phones, is producing large amounts of a new kind of data called trajectories of moving objects. Trajectory data can be useful in several domains such as traffic management, animal behavior analysis, tourism, and marketing, among others. In order to extract information from these data, new methods and algorithms are needed to process large data amounts and make them more meaningful.

A raw trajectory is a sequence of geographic points visited by a moving object during a certain period of time. It can be represented as a sequence of observations of the form (x, y, t), where x and y refer to the position of the moving object, using some coordinate system, and t is the time of the observation. The observations are ordered according to the ascending values of the timestamps t.

Several works have been proposed for trajectory data analysis and mining. These works can be split into two main categories: approaches that work with raw trajectories and approaches that focus on semantic trajectories. This article focuses on the second category. From the semantic point of view, the work of [16] is the most known. It introduced the idea that a trajectory of a moving object is more than a set of points. According to Spaccapietra, a trajectory is a set of important places visited by the moving object, for a given reason and during a minimum amount of time. Thus, a trajectory can be considered as a sequence of stops and moves through a set of interesting sites. An interesting site *IS* is a geographic location that is important for the application (e.g., a hotel, a touristic place). Following this idea, Alvares in [1] proposed the first algorithm to instantiate the model of stops and moves, called SMoT (**S**tops and **M**oves **o**f **T**rajectories). This method is simple, though helpful, to associate meaning to trajectory segments.

An interesting site s can be represented by a 2-tuple $s = (Rs, \Delta s)$, where Rs is a topologically closed polygon in $\mathbb{R}^2$, and Δs is a strictly positive real number that represents the minimum time that a moving object should stay in Rs to consider that this moving object has visited (stopped in) s [1].

Given a set of interesting sites *IS*, the trajectory of a moving object can be enriched with additional information, e.g., the subset of *IS* visited by the object. According to [1], a stop is a portion of a trajectory (i.e., a sequence of consecutive spatio-temporal observations) inside an interesting site $s \in IS$, and that meets the Minimum Staying Condition (MSC). The MSC states that the time covered by the consecutive observations in a stop must be greater than or equal to the threshold Δs, i.e., the moving object must stay inside Rs for a period of time greater than or equal to Δs for this stay to be considered a stop. On the other hand, a move is a portion of a trajectory whose spatio-temporal observations are not contained in any interesting site, or if they are, their covered time does not meet the MSC.

The SMoT algorithm [1] checks the intersection of each trajectory with the set of sites *IS* which may be interesting for the problem at hand (e.g., hotels, cinemas, restaurants, shopping malls). The interesting sites can be collected from a geographical data source (e.g., Google Earth, shape files, geographical databases). Sites intersected by consecutive observations (spatial temporal points) of a trajectory for a minimum amount of time are considered as stops, and the observations in that intersection are labeled as part of the stop in the respective site.

The SMoT algorithm only considers disjoint spatial sites. However, in the geographic space many sites are spatially related to each other. For instance, consider a particular shop, inside a shopping mall, which is located in a particular district, which, by its turn, is located in a city, and so on. As the SMoT algorithm does not consider possible intersections or nesting of the geographic sites of interest, applying it to such a site collection can lead to misinterpretations. For example, suppose that a moving object has passed through a shop s_1 and, after a while, through another shop s_2, both located inside the same shopping mall s_m, which is located in neighbourhood n of a city c. Assume that the trajectory intersections with each of these sites took longer than the respective time threshold to consider a stop in that site. Using SMoT, only the city c may be added to the set of stops of the trajectory. Discovering the stops in each site of the set $\{s_1, s_2, s_m, n, c\}$ would give more information to the analyst, with respect to the meaning of the trajectory. The stops in all nested regions would more realistically express the trajectory semantics, in different levels of the regions nesting hierarchy.

Considering these limitations of the SMoT algorithm, this article introduces a new algorithm, called SMoT+, to find stops in nested sites, i.e., regions contained in other regions. In the example just described, SMoT+ is able to identify stops in the city c, and also in neighbourhoods, shopping malls, specific shops, and other interesting sites presenting or not containing relationships with other sites. Given a set of interesting sites IS (with the possibility of nested sites) and a set of trajectories, SMoT+ can detect stops in any interesting site $s \in IS$, in spite of s being within other sites in IS or not. Similarly to the original SMoT algorithm, SMoT+ traverses each trajectory just once, i.e., it reads each observation point p from each trajectory t and efficiently finds all the (possibly nested) interesting sites that p is within. SMoT+ takes advantage of knowledge about the containment relationships between interesting sites to infer the intersection of a point p with all the interesting sites containing the smallest interesting site that intersects p. Then, it uses this information to calculate the total time that the moving object spends in each interesting site in the upper levels of the containment hierarchy. Thus SMoT+ can efficiently discover all the stops in any site s, provided that the MSC holds for consecutive observations inside s.

Experiments with real data show the effectiveness and the efficiency of the proposed method. Our implementation of SMoT+ has detected stops in nested regions that are not detected by the original SMoT algorithm. Furthermore, our SMoT+ implementation has presented slightly higher processing time than the original SMoT algorithm to discover stops in nested regions. For disjoint interesting sites, the performance loss of SMoT+ due to additional processing to cope with the possibility of nested regions is lower than for processing nested sites.

The rest of this article is organized as follows. Section 2 discusses some related works. Section 3 describes the SMoT+ algorithm. Section 4 reports results of an evaluation of SMoT+ in a case study. Finally, Section 5 concludes the article and suggests directions for future research.

2 RELATED WORKS

Several works have been trying to extract information from trajectories. Works like [6, 7, 8, 10, 11, 12, 15] have developed data mining methods to extract patterns from groups of trajectories. These methods consider the density of the observations to extract behavior patterns from large collections of trajectories.

Another group of works is trying to give more meaning to trajectories [1, 2, 5, 9, 13, 14, 16] considering, apart from space and time information:

1. contextual data such as relevant elements of the geographic space where the trajectories occur,
2. information about the application domain, and
3. metadata that help to enrich trajectory data with more semantic information.

Most of these works deal with patterns of a single trajectory instead of trajectory groups.

Ashbrook in [2], for instance, identifies as stops the portions of a raw trajectory of a car where the signal fails (e.g. the GPS is set off or the car enters a garage) or the speed is zero for a given temporal interval (e.g. the car is parked).

Cao [4, 5] presents an interesting method for finding important places in trajectories. It generates semantic trajectories, considering GPS gaps with missing points that satisfy a minimal time condition. The first step is to find the stay points (stops), which also represent the gaps where the GPS is turned off. The first point of the trajectory after the GPS is turned on and the last point before the GPS is turned off represent the starting and ending point of a stay, respectively. The time duration of the gap must be higher than a given threshold. In the second step, the location of the starting point of the gap is mapped to a street address (using Google Maps). The next step is to search for this address in the yellow pages to give a meaning to the address (e.g. if the address of the trajectory point corresponds to a hotel, this stay point will be labeled with the name of the hotel). According to Cao, the yellow pages service returns a list of semantic locations that are near the queried street address, and one of them may match exactly the given street address of the stay point of the trajectory. The last step is to generate the list of places (locations) visited by the trajectories, based on the duration and the frequency of the visits. The main problem of this approach is that for several applications and depending on the amount of data, the on-line Google mapping is not trivial, and yellow pages information may not be available. In an animal tracking application, for instance, no yellow pages information may be available. This solution is interesting when the trajectory stop can be mapped to a street, but it fails because several important places (stops) may be missed when a trajectory keeps recording the points without generating gaps.

In the method proposed in [9] the user manually annotates trajectories with semantic information, a task that is only feasible for a few trajectories and if the background information is known. In [14], a different approach has been presented to compute stops of trajectories in interesting sites. This algorithm has two main steps. The first step evaluates each single trajectory separately, generating trajectory segments (sub-trajectories) whose average speed is lower than the average speed of the whole trajectory. The second step performs the same process as the SMoT algorithm, for the trajectory segments with average speed below the general average, identified in the previous step. It checks if the trajectory low speed segments intersect disjoint geographic sites. Each portion of a low speed trajectory segment intersecting an interesting site and satisfying the MSC is labeled with the name of the geographic site; otherwise, it is labeled as an unknown stop.

The approach of Baglioni [3] uses an inference engine to automatically classify semantic trajectories with annotated stops and moves into activity categories. Stops and moves are annotated based on a domain ontology. This approach uses the concept of stops, but it does not care about how stops are computed.

More recently, Manso [13] developed an algorithm that computes stops of trajectories based on the variation of the direction of the trajectory. All sub-trajectories of an individual trajectory that have a direction variation higher than a given threshold are considered as stops. Then the SMoT algorithm can be applied to check if such sub-trajectories intersect disjoint geographic sites or not.

The SMoT algorithm [1], which we extend in this article, identifies the stops and moves of a trajectory based on the intersection of the trajectory with a set of interesting geographic places, each one with an associated minimal stay duration to be considered a stop. Hereafter we call this set of interesting sites *IS*. A sub-trajectory must intersect an interesting site of *IS* for a minimal amount of time Δs. The MSC is used by SMoT to determine the stops in these sites. For instance, a hotel will be a stop when the trajectory intersects its location for, say, at least $\Delta s = 5$ hours. As far as we know, the SMoT method is the only one that uses real geographic locations for generating stops if the trajectory intersects a location for a minimal amount of time. We strongly believe that this is the best way to give real meaning to trajectories.

Nevertheless, one main problem with SMoT is that it only generates stops for *disjoint* geographic sites. In case there are several nested interesting sites, what is very common in the real world (e.g. cities contain districts, districts contains streets, districts are crossed by rivers, districts may contain schools, hospitals and so on), only the first discovered site is recorded. For instance, if SMoT discovers that a street is a stop, it will not look to which district and city this street belongs to, or, if there is a stop at a university, SMoT will not discover in which rooms or places of the university that stop is located. The following section details our proposed extension of SMoT to cope with nested interesting sites.

3 SMOT+: AN EXTENSION OF THE SMOT ALGORITHM

This section presents the SMoT+ algorithm, an extension of SMoT that generates all stops in nested interesting sites. Let us consider the example shown in Figure 1 where, for instance, the interesting site s_4 is inside s_3, which in turn is inside s_1.

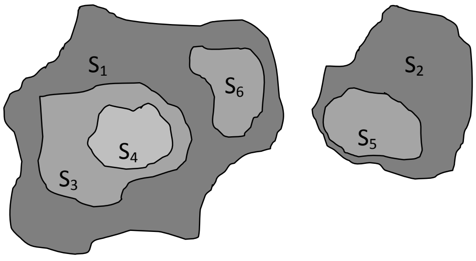

Figure 1. A set of non-disjoint interesting sites with nesting relationships

Similarly to SMoT, SMoT+ identifies the stops and moves of each trajectory T, by traversing T just once. SMoT+ stores temporal information associated to every interesting site in a set of interesting sites *IS* in an array called *dataNodeArr*, which helps to identify the stops. Each element in *dataNodeArr* is an ordered pair (*startObs*, *endObs*), where *startObs* is the sequential number of the observation when T went inside an interesting site s, and *endObs* is the number of the observation when T went outside s.

The algorithm SMoT+ uses as input a knowledge base, representing a hierarchy H, of containment relationships among the interesting sites from the set *IS*. Figure 2 shows the corresponding hierarchy of containment relationships for the sites of Figure 1 (e.g. s_1, for instance, at level 1 of H, contains sites s_3, s_6 and s_4). Note that the dummy site with $id = 0$ (s_0) on the hierarchy top covers the whole geographic space, and consequently all sites from the set IS.

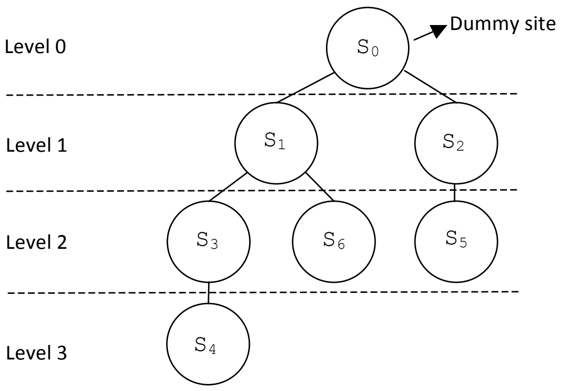

Figure 2. Hierarchy of containment relationships among sites shown in Figure 1

Listing 1 presents the SMoT+ algorithm, which works as follows. Each trajectory T is traversed from its first to its last observation using variable i (line 9). Variable *currentSIS* keeps the smallest site intersected by the current observation of T. It is initially set to zero (the dummy interesting site covering all the other sites) (line 7). The set *currentIntersectedIS* keeps the subset of identifiers of interesting sites intersected by the current observation of T, and *previousIntersectedIS* represents the subset of identifiers of interesting sites intersected by the previous observation of T.

When a trajectory intersects one or several sites (subset *currentIntersectedIS*), the variable *smallestIntersectedSite* is assigned the identifier of the most nested (smallest) site in the set *currentIntersectedIS* (line 11). If the value of *smallestIntersectedSite* is different from *currentSIS* then *currentSIS* is assigned the value of *smallestIntersectedSite* (lines 12 and 13), and the following actions are performed:

- For each interesting site in *IS* (lines 14–16), where the trajectory has just gone inside (set *currentIntersectedIS–previousIntersectedIS*), the attribute *startObs* (of its *dataNode*) stores the number of the current observation of the trajectory (line 15).
- For each interesting site in *IS* (lines 17–24), where the trajectory has just gone outside (set *previousIntersectedIS–currentIntersectedIS*), the attribute *endObs* (of its *dataNode*) stores the number of the previous observation of the trajectory (line 18). Next, the algorithm finds the staying time in each interesting site, by computing the difference between the time of *endObs* and the time of *startObs* (line 19). The algorithm checks if this time is greater than or equal to the staying time threshold of the interesting site (line 20), and if so, the algorithm has found a stop (lines 21–22).

```
ALGORITHM SMoT+
INPUT:   $\mathcal{T}$ // Set of trajectories
         $\mathcal{IS}$ // Set of interesting sites $\{s_1 = (Rs_1, \Delta s_1), \ldots, s_n = (Rs_n, \Delta s_n)\}$
         $\mathcal{H}$ // Hierarchy of containments among the sites from IS
OUTPUT:  $\mathcal{S}$ // Set of stops
         $\mathcal{M}$ // Set of moves
1. BEGIN
2. $\mathcal{S} = \phi$; $\mathcal{M} = \phi$;
3. dataNodeArr[0, n]; // Array of $n+1$ dataNodes
4. currentIntersectedIS = $\phi$; // Set of site ids
5. previousIntersectedIS = $\phi$; // Set of site ids
6. FOR each trajectory $T \in \mathcal{T}$ LOOP
7.   currentSIS = 0; // Smallest site intersected by the current obs.
8.   $i = 1$; // First obs. of T
9.   WHILE ($i \leq size(T)$) DO // Traverse T
       /* Find the set of site ids intersected by point(T[i]),
       point function extracts the coordinates x and y of T[i] */
10.    currentIntersectedIS = intersectedIS(point(T[i]), H);
11.    smallestIntersectedSite = id of the smallest intersected
                                  site in currentIntersectedIS;
12.    IF smallestIntersectedSite $\neq$ currentSIS THEN
```

```
13.       currentSIS = smallestIntersectedSite;
          // Sites where obs. i of T just went inside
14.       FOR each siteId ∈ (currentIntersectedIS − previousIntersectedIS) LOOP
15.         dataNodeArr[siteId].startObs = i;
16.       END FOR
          // Sites where obs. i of T just went outside
17.       FOR each siteId ∈ (previousIntersectedIS − currentIntersectedIS) LOOP
18.         dataNodeArr[siteId].endObs = i − 1;
19.         elapsedtime = time(T[dataNodeArr[siteId].endObs])
               − time(T[dataNodeArr[siteId].startObs]);
20.         IF (elapsedtime ≥ $\Delta s_{siteId}$) THEN // A stop of T in siteId was found
21.           Stop = (T, siteId, time(T[dataNodeArr[siteId].startObs]), time(T[i − 1]));
22.           S.add(Stop);
23.         END IF
24.       END FOR
25.     END IF
26.     previousIntersectedIS = currentIntersectedIS;
27.     i++; // Next obs. of T
28.   END WHILE
29.   Add to M a new move with each maximal set of consecutive
        observations of T that are not part of any stop.
30. END FOR
31. END
```

Listing 1. The SMoT+ algorithm

```
   FUNCTION intersectedIS(p,H)
INPUT:    p // Spatial point (x, y)
OUTPUT: IntersectedIS // ids of interesting sites intersected by p
1. BEGIN
2. found = FALSE;
3. currentLevel = H.numberOfLevels(); // Height of H
4.   WHILE NOT found DO
5.     FOR each siteId ∈ H||H.level(siteId) = currentLevelLOOP
6.       IF (p intersects $Rs_{siteId}$) THEN
7.         found = TRUE;
8.         BREAK; // Exit FOR
9.       END IF;
10.    END FOR;
11.    currentLevel–; // Go to the next level of H
12.  END WHILE;
13. RETURN {siteId} ∪ H.ancestors(siteId);
14. END
```

Listing 2. Function intersectedIS

The function *intersectedIS*, see Listing 2, returns the identifiers of the interesting sites (subset *currentIntersectedIS*) that a trajectory intersects with its point p of its current observation. It begins by searching for the most nested sites in H, located in its deepest level (line 5). Whenever there is no site intersecting the point p in a level, it proceeds to the immediate upper level (line 11). When it finds a site intersected by p (line 6), the function immediately returns the identifier of this site along with the identifiers of all of its ancestors (line 13). In this way, the function avoids unnecesary checking for intersections. For example, consider observation 5 in Figure 4. As the smallest intersected site by observation 5 is s_3, the function returns the identifiers of the intersected sites $\{3, 1, 0\}$, as shown in Figure 3.

Note that if a trajectory point does not intersect any region, it still intersects the dummy site (with $id = 0$) and thus the function *intersectedIS* returns $\{0\}$.

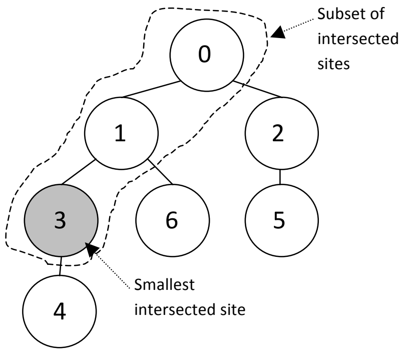

Figure 3. Subset of intersected sites by observation 5 of the trajectory of Figure 4

4 SMOT+: A RUNNING EXAMPLE

Let us consider the set of interesting sites $IS = \{(s_1, 10\,\text{min}), (s_2, 20\,\text{min}), (s_3, 20\,\text{min}), (s_4, 10\,\text{min}), (s_5, 5\,\text{min}), (s_6, 10\,\text{min})\}$, and a trajectory T with observations at every 5 minutes, starting at observation 1 and ending in observation 29, as shown in Figure 4.

The SMoT+ algorithm begins with observation 1 and proceeds to observations 2 and 3. When it finds out that T intersects s_1 at observation 3, it makes the following assignments *currentIntersectedIS* $= \{1\}$, *smallestIntersectedSite* $= 1$, and *currentSIS* $= 1$. The attribute *dataNode.startObs* for site 1 is set to the number of the current observation (3), and *previousIntersectedIS* $=$ *currentIntersectedIS* $= \{1\}$. When the algorithm reaches observation 4, *smallestIntersectedSite* does not change, but when it reaches observation 5, the following assignments occur: *currentIntersectedIS* $= \{1, 3\}$ and *smallestIntersectedSite* $= 3$. Then, because *smallestIntersectedSite* $\neq$ *currentSIS*, SMoT+ stores the number of the current observation (5) in each site where T just entered, i.e., *currentIntersectedIS* $-$ *previousIntersectedIS* $= \{1, 3\} - \{1\} = \{3\}$. In addition, because *previousIntersectedIS* $-$ *currentIntersectedIS* $= \{3\} - \{1, 3\} = \phi$, we know that T has not gone outside of any site.

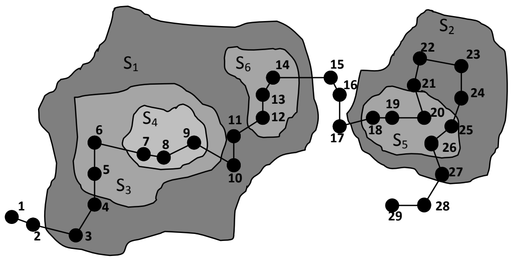

Figure 4. A set of non-disjoint interesting sites and a trajectory

When the algorithm reaches observation 7, *currentIntersectedIS* $= \{1, 3, 4\}$. Therefore, T just went inside the site s_4. The attribute *dataNode.startObs* of this site is set to the number of the current observation (7). At observations 8 and 9, T has not entered new sites and has not gone outside any site.

Now when the algorithm reaches observation 10, *currentIntersectedIS* $= \{1\}$ and *previousIntersectedIS* $= \{1, 3, 4\}$. Then, because *currentIntersectedIS* $-$ *previousIntersectedIS* $= \phi$, SMoT+ detects that T has not gone inside other sites. However, because *previousIntersectedIS* $-$ *currentIntersectedIS* $= \{1, 3, 4\} - \{1\} = \{3, 4\}$ (sites from which the trajectory has just gone outside); SMoT+ stores the number of the previous observation (9) in each of the sites $\{3, 4\}$ and checks stops in these sites.

Table 1 summarises the results obtained by applying SMoT and SMoT+ to the trajectories and the sites presented in Figure 4 with the minimum staying times specified in *IS*. Note that only SMoT+ detected the stops in the nested sites s_3, s_4, s_5, and s_6, shown in gray in Table 1.

SMoT			**SMoT+**		
Observation	**Type**	**Interesting site**	**Observation**	**Type**	**Interesting site**
(3–14)	Stop	s_1	(3–14)	Stop	s_1
(18–27)	Stop	s_2	(5–9)	Stop	s_3
			(7–9)	Stop	s_4
			(12–14)	Stop	s_6
			(18–20)	Stop	s_5
			(25–26)	Stop	s_5
			(18-27)	Stop	s_2
(1–3)	Move		(1–3)	Move	
(14–18)	Move		(14–18)	Move	
(27–29)	Move		(27–29)	Move	

Table 1. Results of SMoT and SMoT+

5 EXPERIMENTS

Experiments were performed on a set of a 100 real GPS trajectories of vehicles, collected in the city of Rio de Janeiro. The total number of observations of the trajectories was 268,900. The trajectories cross four regions called localities (Madureira, Mier, Realengo, and Vila Isabel) and 137 neighbourhoods of Rio de Janeiro. A total of 57 neighbourhoods are inside (nested neighbourhoods) of the four localities, so we have four sites with several smaller regions inside, and 80 are outside (non-nested neighbourhoods). Figure 5 shows a map of Rio de Janeiro and two trajectories. The big regions represent the four localities and the smaller regions are the nested sites (nested neighbourhoods).

In summary we have the following hierarchy: level 0 – Rio de Janeiro city, level 1 – the four localities, and level 3 – the neighbourhoods.

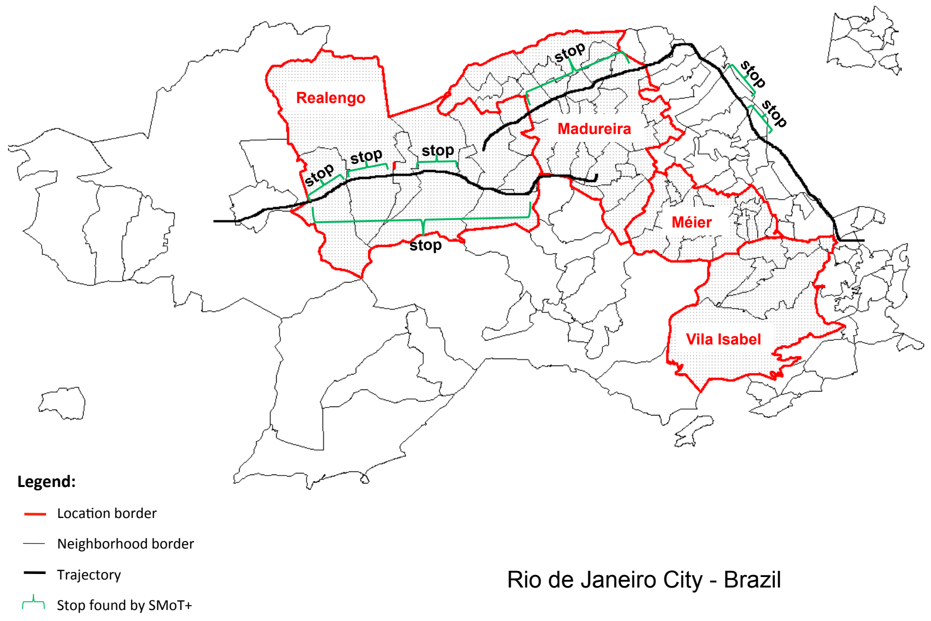

Figure 5. Stops discovered by SMoT+ in four of Rio's localities and their nested neighbourhoods crossed by two trajectories

Both algorithms were tested with a minimal duration of 7 and 10 minutes for a region to be considered an interesting site. Figure 6 summarizes the results. Note that the number of stops identified with both algorithms decreases as the threshold increases. The SMoT algorithm identified much less stops than SMoT+, and the reason is because SMoT cannot identify stops in nested neighbourhoods. Indeed, by applying the SMoT algorithm with a threshold of 7 minutes, stops were only identified in the four localities and in the 80 non-nested neigborhoods, i.e., SMoT was unable to identify the 19 stops that occurred in the 57 nested neighbourhoods and that were identified by SMoT+. In a similar way, by applying the SMoT algorithm with a threshold of 10 minutes, it only identified 43 stops and was unable to identify the 10 stops that occurred in the 57 nested neighbourhoods.

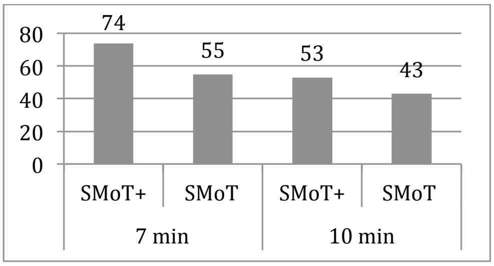

Figure 6. Summary of identified stops with minimal times: 7 minutes and 10 minutes

Figure 7 shows the running time of both algorithms considering the dataset with nested sites. The running time of SMoT+ is a little higher because of the set operations and the exploration of the hierarchy of containments.

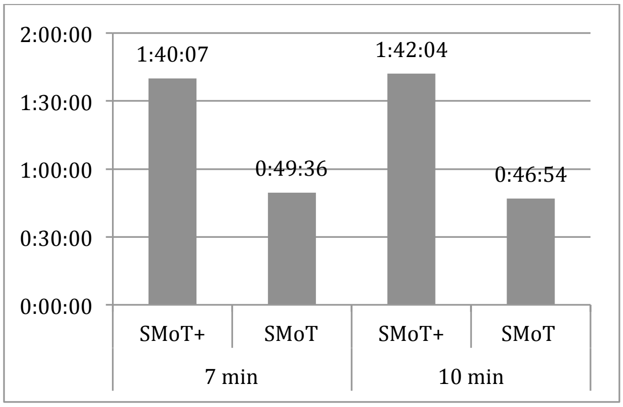

Figure 7. Running time of SMoT and SMoT+ with nested IS

Figure 8 shows an experiment of SMoT and SMoT+ on a dataset without nested sites. In this case, the performance of SMoT in relation to SMoT+ is better, but the time difference is not so high. Although we show the running time comparison, the objective of this article is not on proposing a more efficient method, but an algorithm that finds better results, adding more semantics to trajectories. Performance issues is a general problem in trajectory data analysis, but so far, trajectory data analysis and mining proposals have not focused on efficiency, but rather on efficacy of the discovery methods and more meaningful patterns.

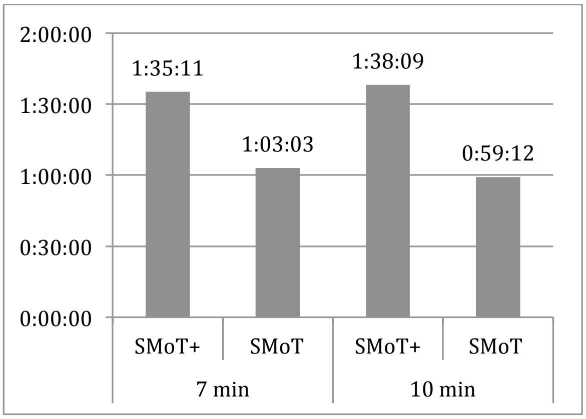

Figure 8. Running time of SMoT and SMoT+ without nested sites

6 CONCLUSIONS AND FUTURE WORK

Semantic enrichment of trajectories is becoming a very important issue, since the information contained in raw trajectories is limited and difficult to interpret. In addition, real trajectory data can be related to geographical features, such as sites (e.g., regions) of interest. Trajectories can cross and/or stop within these sites for some amount of time, and this information can be crucial for understanding trajectories. However, only a few works in the literature consider the integration of trajectories with geographic information.

The well-known SMoT algorithm is one of the few ones proposed by now to consider geographic information for trajectory analysis. However, a major problem of SMoT is that it considers that a trajectory may intersect only disjoint geographic regions, while in the real world many geographic regions are nested or present some topological relation with other regions (contains, inside, overlaps, etc).

In this paper, we presented an extension of the SMoT algorithm, called SMoT+, to discover stops of trajectories in nested geographic sites. SMoT+ is able to identify all stops in these sites, even those stops that share observation points that intersect distinct nested regions. In a similar way, as the original SMoT algorithm, the SMoT+ time traverses each trajectory and considers each trajectory observation point only once. In addition, SMoT+ uses knowledge about the containment hierarchy among interesting sites to efficiently determine all the interesting sites that intersect each trajectory point, by starting with the smallest sites in the hierarchy, and inferring intersections with all sites containing a smaller intersected site. Thus, it can generate stops within any site, including nested sites, while minimizing the number of computations of intersections between observation points and interesting sites.

Experiments with real data and a running prototype of SMoT+ have shown that it finds more detailed results than the original SMoT algorithm, by discovering stops in nested sites. Therefore it contributes more to the semantic enrichment and understanding of trajectories in different levels of the geographic regions hierarchy. Furthermore, its running time is just lightly higher than that of the original SMoT algorithm.

As future works we plan the following directions:

- Analysing semantic trajectories in different granularity levels: SMoT+ enables semantic trajectory representation in different spatial granularities. Thus, it can be used, for example, to build trajectories data warehouses that allow drill-down to more detailed trajectory representations, by considering stops in finer grained regions.
- Discovering of proximities: a trajectory might pass near interesting sites without going inside them. For example, a tourist might pass near a hotel or a restaurant, a vehicle might pass near a dangerous zone. The detection of these proximities could help to attract more customers or to prevent accidents.
- Identification of non-continuos stops: a trajectory can satisfy the minimal time condition in an interesting site with some gaps. For example, consider a shop with a MSC of 30 minutes. Suppose that a customer enters the shop, stays 20 minutes. Then s/he goes out and 5 minutes later s/he comes back and stays for another 20 minutes. Thus, the customer stayed in the shop for 40 minutes in total, but with a gap of 5 minutes.

Acknowledgments

The authors would like to thank CNPQ that partially provided support for this research.

REFERENCES

[1] Alvares, L. O.—Bogorny, V.—Kuijpers, B.—de Macedo, J. A.—Moelans, B.—Vaisman, A.: A Model for Enriching Trajectories With Semantic Geographical Information. In Proceedings ACM-GIS, ACM Press 2007, pp. 162–169.

[2] Ashbrook, D.—Starner, T.: Using GPS to Learn Significant Locations and Predict Movement Across Multiple Users. Personal Ubiquitous Computing, Vol. 7, 2003, pp. 275–286.

[3] Baglioni, M.—de Macedo, J. A.—Renso, C.—Trasarti, R.—Wachowicz, M.: Towards Semantic Interpretation of Movement Behavior. AG-ILE Conference 2009, pp. 271–288.

[4] Cao, H.—Mamoulis, N.—Cheung, D. W.: Discovery of Periodic Patterns in Spatiotemporal Sequences. IEEE Transactions on Knowledge and Data Engineering, 2007, pp. 453–467.

[5] Cao, X.—Cong, G.—Jensen, S. C.: Mining Significant Semantic Locations from GPS Data. 2010, pp. 1009–1020.

[6] Djordjevic, B.—Gudmundsson, J.—Pham, A.—Wolle, T.: Detecting Regular Visit Patterns. Algorithmica 2011, pp. 829–852.

[7] Dodge, S.—Weibel, R.—Laube, P.: Exploring Movement-Similarity Analysis of Moving Objects. SIGSPATIAL Special 2009, pp. 11–16.

[8] Giannotti, F.—Nanni, M.—Pinelli, F.—Pedreschi, D.: Trajectory Pattern Mining. ACM 2007, pp. 330–339.

[9] Guc, B.—May, M.—Saygin, Y.—Korner, C.: Semantic Annotation of GPS Trajectories. Conference on Geographic Information Science, Gerona 2008.

[10] Laube, P.—van Kreveld, M.—Imfeld, S.: 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] Lee, J. G.—Han, J.—Whang, K. Y.: Trajectory Clustering: A Partition-and-Group Framework. ACM Conference 2007, pp. 593–604.

[12] Lee, J. G.—Han, J.—Li, X.—Gonzalez, H.: TraClass: Trajectory Classification Using Hierarchical Region-Based and Trajectory-Based Clustering, Conference, VLDB '08, 2008, pp. 779–790.

[13] Manso, J. A.—Times, V. C.—Oliveira, G.—Alvares, L. O.—Bogorny, V.: A Direction-Based Spatio-Temporal Clustering Method. IEEE, Conference on intelligent systems, 2010.

[14] Palma, A. T.—Bogorny, V.—Kuijpers, B.—Alvares, L. O.: A Clustering-Based Approach for Discovering Interesting Places in Trajectories. ACM Press, New York 2008, pp. 863–868.

[15] Sakr, M.—Andrienko, G.—Behr, T.—Andrienko, N.—Hartmut, R.—Hurter, C.: Exploring Spatiotemporal Patterns by Integrating Visual Analytics With a Moving Objects Database System. Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 2011, pp. 505–508.

[16] Spaccapietra, S.—Parent, C.—Damiani, M. L.—Macedo, J. A.—Porto, F.—Vangenot, C.: A Conceptual View on Trajectories. Data & Knowledge Engineering, 2008, pp. 126–146.

Francisco Javier Moreno Arboleda holds a Ph. D. and a M. Sc. in Engineering Systems from Universidad Nacional de Colombia. He currently works as an associate professor at Universidad Nacional de Colombia, sede Medellín. His research areas are datawarehouses and spatiotemporal databases.

Andrés Pineda holds a degree in Engineering Systems from Universidad Nacional de Colombia. He currently works as an independant consultant and is finishing his M. Sc. studies at Universidad Nacional de Colombia, sede Medellín. His research areas are algorithms and spatiotemporal databases.

Renato Fileto holds Ph. D. and M. Sc. degrees in computer science from Campinas State University (1992–1994, 1999–2003), with internship at Georgia Institute of Technology (2002), and a Bachelor degree in computer science from Federal University of Uberlândia (1988–1992). His research carrier has been intertwined with activities in the industry. Since 2006, he is a Permanent Professor at the Department of Informatics and Statistics of Santa Catarina Federal University, Florianópolis, Brazil. He is the author of more than 50 publications and has been reviewer for several conferences. His research area covers databases, with focus in complex data indexing and retrieval, data semantics, spatiotemporal data warehousing, and workflows for data integration and information analysis.

Vania Bogorny is professor at Departamento de Informática e Estatística 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. During the Ph. D. program she was visitor scholar at University of Minnesota, USA (September/2004 to March/2005), with Prof. Shashi Shekhar's group. From November 2006 to January 2008 she joined the research staff of the Theoretical Computer Science Group, of Hasselt University, Belgium, to work in the context of the European project GeoPKDD. In 2007, she received the Best Ph. D. thesis Award from the Brazilian Computer Society. She has published 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), Geoinformática, Transactions in GIS and International Journal of Geographical Information Science (IJGIS). She has served as 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.