A Problem of Detecting Stops While Tracking Moving Objects Under the Stream Processing Regime

Paweł M. Białoń

National Institute of Telecommunications, Warsaw, Poland

https://doi.org/10.26636/jtit.2023.4.1481

Abstract — The tracking of moving objects with the use of GPS/GNSS or other techniques is relied upon in numerous applications, from health monitoring and physical activity support, to social investigations to detection of fraud in transportation. While monitoring movement, a common subtask consists in determining the object's moving periods, and its immobility periods. In this paper, we isolate the mathematical problem of automatic detection of a stop of tracking objects under the stream processing regime (ideal data processing algorithm regime) in which one is allowed to use only a constant amount of memory, while the stream of GNSS positions of the tracked object increases in size. We propose an approximation scheme of the stop detection problem based on the fuzziness in the approximation of noise level related to the position reported by GNSS. We provide a solving algorithm that determines some upper bounds for the problem's complexity. We also provide an experimental illustration of the problem at hand.

Keywords — algorithm complexity, GPS monitoring, object tracking, physical activity monitoring, stream processing

1. Introduction

The tracking the geographical position of various objects has become a ubiquitous technique due to the proliferation of global positioning systems (GPS), global navigation satellite systems (GNSS) and other location solutions. The spectrum of tracked objects is rather broad. These may include, for instance, humans, vehicles, commodities or electronic equipment. A bulk of research seems to concentrate on monitoring people and vehicles. The monitoring of individuals serves several purposes. Healthcare is the most important area here, as the physical activity of individuals may be controlled – see [1] for a survey. The monitoring of the locations of people serves also a more general purpose, i.e. identifying their "life patterns" – the routes they usually cover and, importantly for us, the indoor and outdoor places they usually visit or at which they stay for a certain period of time – see [2] and the references therein. This feature is harnessed to offer targeted advertisements and recommendations, to provide personalized assistance in schedule and route planning, to investigate various social trends and, once again, to protect the health of humans, as the monitoring of adolescents staying at locations in which they may smoke offers the potential to mitigate that specific health risk [3]. Vehicle monitoring [4] is another important area of tracking physical objects. It is relied upon by transportation companies and, sometimes, by government authorities willing to track some sensitive commodities carried over the road. The main purpose of vehicle monitoring is to prevent driver fraud. Such fraud may consist in illegally loading or dropping commodities off, as well as in picking extra passengers by taxi drivers. Other objectives may be achieved as well, since transportation companies are able to observe the progress of deliveries and monitor the personal safety of drivers traveling to remote areas. Compliance with traffic rules and working time requirements may be controlled as well.

Traccar [5] is a large open-source vehicle monitoring system. It consists of tools used for collecting position data from various on-board devices and a monitoring server. It is capable of visualizing the positions and routes of specific vehicles on a map and offered several automated analyses focusing on the routes taken and potential fraud. The range of techniques used to automatically detect driver fraud is rather extensive. Geofencing [6] is a simple technique used for observing whether a given vehicle leaves the corridors predefined for a given route. Many advanced approaches are reported, like using the Dempster-Shafer evidence theory [7], support vector machines [8], clustering of routes [9], to name just a few.

A common subproblem encountered in automated monitoring of objects seems to consists in discovering when a given object is moving and when it is immobile. It is known as the so-called "stop detection problem", where we need to decide whether the object experienced a period of immobility given the sequence of its GNSS frames. Each frame consists of a single observation of the object, namely the time of the observation and the coordinates describing its position. We could optionally demand that the algorithm returns the position at which the object stops and the duration of its immobility period. When a human being stops, they usually perform no physical activity. They are also likely to stay at a predefined location, such as a flat, a restaurant, an office, a shop, etc. Traccar, similarly to many other vehicle monitoring systems, displays moving vehicles as green points on the map, and stopped vehicles are identified by red points. Stop detection techniques must be applied in order for such a simple signaling system to operate. Complex fraud detection algorithms examining the shape of the vehicle's route and its deviation from the normal pattern could also benefit

from stop-related information. The process of picking up or dropping off illegal commodities requires the vehicle to stop, hence the need for accurate stop detection techniques.

2. Literature on Stop Detection

We must be aware that the problem of stop detection is often not verbalized by the authors of object tracking systems, not speaking about revealing its solution method. Perhaps it is considered an obvious part of a much more complicated software. Scientific literature provides a more generous treatment of this problem, albeit it somewhat selectively concentrates on specific solution approaches.

The problem of stop detection has various definitions. However, one of them is ubiquitous – having a sequence of GNSS positions and the sequence of corresponding GNSS times, a stop (stay) of an object is defined there as staying within a circle of the given radius at least for the given time. This definition is very natural. Small movements during the stay, within the reach of the radius, are allowed to reflect either the noise in the reported GNSS position in the samples, or movements of the object within a place like shop or work office. This definition is used in [2], [10]–[12]. Others [13], [14], rather treat the problem as a recognition task for a machine, do not define it precisely, and compare the recognition of the algorithm with the human supervisor statements about when the stops actually took place.

The literature stop detection methods will be to a great extent analyzed according to their time and memory effectiveness. For this sake we shall use a notion of *stream processing* regime.

Stream processing is a setting when an input to the algorithm is a possibly infinite stream of consecutive objects, say, of the same type, like GNSS frames, each containing time and the corresponding object position. The algorithm is fed the consecutive elements of the stream. The algorithm itself can use only a limited memory. This limit is constant in the number of stream elements seen so far. Hence, the algorithm cannot even store the whole so-far seen trajectory of the object in its data structures. Instead, it must use some aggregates.

The stream processing setting is important in many object tracking applications, mainly due to the massive character of many such systems. A fleet monitoring systems of a large petrol company, or some government car monitoring system may watch hundreds of thousands of cars. The cars movements must be analyzed simultaneously and on-line. Hence, it creates efficiency problems for the computing servers, both in terms of used memory and processor power. Stream algorithms clearly help to decrease the memory requirements. They are also potentially faster, since they are by far differently constructed than their non-stream counterparts, for example, they do not iterate over a large array that stores the fragment of input streams seen so far.

Stop detection methods might be of various complication level. Simple, technical, easily implementable approaches are valuable in practical monitoring systems, or monitoring smartphone applications. They traditionally use a wider set of input data, not only GPS positions and timestamps. Such data might be the information whether the car engine is on or off, which highly correlates with stops. Another data might be object speed taken from GNSS. The authors of [13] use simple data like speed and their straightforward derivatives (rapid speed change, differences in consecutive GNSS timestamps). A predefined decision tree equipped with comparisons of these quantities with predefined thresholds) are used to recognize stops.

Such methods are very fast, usually they are stream algorithms, but it is inherently difficult to reason about their recognition precision, which can be just assessed experimentally. Also, a wider set of data is not always available for particular objects, which may prevent some massive applications.

A stop is visually a concentration of the trajectory points. Thus one naturally tends to use some clustering methods. Various authors have noticed that methods with an a priori given number of cluster, like k-means, are not suitable, since we do not know the number of stops in advance. Giving a too big number would make the algorithm find stops where the object is moving, but slowly. Hence, methods with unspecified number of clusters are used, and the famous DBSCAN clustering method has found an enormous interest – see [10], [14], for instance. This method must be modified for our purpose. Constraints defining cluster are changed to reflect time dependencies: we expect a cluster "duration" to be over a given threshold, and also the cluster must be formed by consecutive frames.

Interestingly, in [14], also entropy of movement direction is used to enhance DBSCAN: during stays, the direction of position movement is chaotic, whereas on a real way it becomes more ordered. An important feature of [14] is using support vector machines to further divide stop places into actual activity stops and non-activity stops (e.g. waiting for a green light). In turn, [10] provides an interesting notion of *common stay points*, when it is required that many monitored objects stay within a given distance of such a point at least for a given time (finding this is not equivalent to correlating ordinary stay points of several objects). Many stay places: shops, parks, etc. are common. Also, the authors consider a hierarchy of stay point (e.g. a university campus might be divided into particular buildings). DBSCAN might be a quite fast method, with linear-logarithmic time complexity, however, its memory complexity is linear instead of constant, which makes derivative approaches not stream. Also, the approaches of [10], [14] remain heuristics. The authors do not strictly prove the correctness of the solution, even when the problem itself is sufficiently strictly defined.

Many other authors form the clusters of points, perhaps simpler that the approaches above. The complexity of such approaches is often large, for example, the authors of [15], [16] use two nested "for" loops to iterate over the set of GNSS probes, thus obtaining an algorithm quadratic in their number, which might suitable only for medium size post-factum analyses.

Two approaches – [2], [12] – are close to the proposition in this paper. They solve the most common stop detection prob-

lem, i.e. staying within a ball of a given radius ρ at least for the given time ϑ. They consume GNSS probes consecutively, as these become available. The probes contain positions and time stamps. They algorithms maintain a sort of "current potential cluster", represented by its potential center, as imagined by the particular algorithm. When the next probe comes, it is decided whether it lies within the cluster (within the distance ρ from the cluster center) and can be added to the cluster, or not – and the cluster should be forgotten and a new cluster created around the new point. Whenever a cluster duration in time exceeds ϑ, a stop is detected. This scheme is extended in [2] by delaying the cluster closing: two points sufficiently distant from it are necessary instead of one point. In [11], in turn the cluster center is not taken from a particular GNSS position but calculated as an arithmetic average of the so-far seen cluster points.

The two algorithms are not precisely stream algorithms. However, they seem quite easily extendable to stream algorithms, because they essentially use low memory, mainly for representing the center of the current potential cluster. The necessary extensions would contain correcting the representation of cluster, combined with a fast computation of cluster duration in time, or a fast calculation of the arithmetic average.

Again, the authors do not provide any strict proof that their algorithms actually solve the problem. As exact solving of many mathematical problems by stream algorithms is not possible, we might expect the same in this case. Perhaps the authors suspected that their solution are not certain and thus introduced their improvements: the delay of cluster change and the representation of cluster center by an arithmetic average of some points (the latter is especially suspicious: since the clusters lie within a ball around some point, this point is rather more close to the distance center of the cluster of points than to the arithmetic average of the points).

3. The Contribution

We propose a stream, and thus efficient, algorithm to solve the stop detection method, of a provable solution quality. Both the stream character and provability of the algorithm which are so scarce in the literature on stop detection. We use only position data and timestamps. We pose a stop (stay) detection problem. It is a variant of the most ubiquitous demand of staying in a ball of the given radius at least by the given time, defined in the stream processing regime. We give an interesting definition of approximation of this problem, based on the fuzziness of the radius defining the problem. We prove that our algorithm gives solutions approximate in this sense.

The algorithm is based on a more complicated design of potential stop clusters, containing many points, but still using a little memory. Our intention is to promote a more formal analysis of the stop problem in the stream processing regime and initiate constructing various suitable provable algorithms. Also, our research has already shown the multitude of definition and solving nuances that appear when we tread the problem formally, like ambiguities of the "stop cluster" and its time limits, problems with overlapping stops. These nuances also deserve investigation.

The rest of the article is structured as follows. In Section 4, we describe the stream processing regime. The stop problem is posed in Section 5. Section 6 gives an approximation solving algorithm, with the discussion of its properties, including this approximate character, defined in a specific way. Finally, some illustrations of solving the problem is given in Section 7, and the work is concluded and directions for a further research are presented in Section 8.

Note. By $\mathrm{dist}(a, b)$ where $x, y \in \mathbb{R}^n$ the Euclidean distance in this space is denoted:

$$\mathrm{dist}(a, b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2}\,.$$

4. Stream Processing Regime

Problems and their solving algorithms might be defined with the use of a stream processing regime, with its properties being as follows:

1) input data has the form of an infinite sequence of identically-typed elements; the algorithm is provided with consecutive elements from the stream; each element is presented to the algorithm only once,
2) the memory of the algorithms is constant in the number of elements presented to the algorithm so far.

"Stream algorithm" is a new name for the former notion of "ideal data processing algorithm" [17]. More precisely, let $(d_1, d_2, \ldots)$, where $d_i \in D$ be a possibly infinite sequence of input data elements, each of the type represented by some set D. The stream algorithm is a box with a constant memory and works in the stream regime (Algorithm 1), where the steps are performed for each element of the sequence $(d_1, d_2, \ldots)$. In practice, algorithms can only partially be of the stream variety, i.e., only a specific portion of the data may form a stream and the constancy of memory may be required only with respect to this part of data. The remaining part of data may be treated in a standard way. It might be fed to the algorithm at the beginning of the computations and the memory complexity of the algorithm with respect to this part of data may be over-constant.

Algorithm 1 Stream processing scheme.

1: Initialize the algorithm memory (state).
2: Consume d_1, perform some processing using the box memory and possibly produce some output.
3: Consume d_2, perform some processing using the box memory and possibly produce some output.
...

5. The Stream Stop Detection Problem

Although the main idea of defining a stop (stay) is simple, there are many nuances and singularities connected with the ambiguity of the time limits applicable to the stay period,

difficulties in crisply defining consecutive stays, potential to represent one stay as two consecutive stays, etc. These nuances are even more pronounced in the stream regime, leading to unavoidable solution inaccuracy. Therefore, some reasoning may seem to be a little confusing. We shall try to simplify the problem posing even beneath the practical application needs, in order to simplify the analyze with outlook to practical needs.

Let the input stream of data be $(d_1, d_2, \ldots)$ where $d_i = (t_i, p_i)$ $t_i \in \mathbb{R}_+$ represents the GNSS time for probe i and $p_i = (x_i, y_i) \in \mathbb{R}^2$ represents the position of the object in probe i.

To describe the position of the probes, a Cartesian plane is used, with an orthonormal system of coordinates and the Euclidean distance. In this way the curvature of the Earth may be ignored, as the problem of stop is local in its nature and may be approximated by adopting such an approach. For big system trajectories representing long vehicle routes, a local transformation of the geographic coordinates (longitude and latitude) into our plane might be recomputed from time to time.

For simplicity, the fact that multiple stops may occur along the object's trajectory is also ignored, as the task is to detect the existence of a stop condition only once, as soon as it occurs. However, the processing may be easily extended to the realistic case of multiple stays, e.g., by resetting the algorithm's state when it is discovered that the stop (stay) has come to an end. Now the problem may be formulated in the following manner:

$$P(\delta_p, \delta_t)\ .$$

The problem-related parameters include time threshold $\delta_t > 0$ and distance threshold $\delta_p > 0$.

In a stream regime, if for some step e of the scheme from Algorithm 1 there exists $b \in 1 \ldots, i,\ e > b, b \in 1 \ldots, i$, such that the following stop condition is fulfilled:

$$t_e - t_b > \delta_t \wedge \exists_{s \in \mathbb{R}^2} \forall_{j=b,\ldots e}\ \ \mathrm{dist}(p_j, s) \leqslant \delta_p\ , \qquad (1)$$

the algorithm must output a "stop" at step e of the scheme from Algorithm 1 at the latest.

The algorithm seeks for a situation when there exists a stable position s, such that the consecutive points are located on a fragment of the trajectory, starting at GNSS probe b, ending with probe e and lasting for at least time δ_t, and the GNSS positions of that fragment are located not more than δ_p away from s (Fig. 1). The use of a lower time threshold while defining a stop is natural. More explanations are required in relation to the position threshold δ_p. It may be interpreted in two ways. For tracking people staying at a specific location, such as a room, or a shop, we must allow for their movement within this facility. So, the coordinates cannot be identical throughout the entire stay. When monitoring vehicles, a vehicle may be staying at the same place, but its GNSS positions may vary around the true position, due to GNSS inaccuracy (noise). The latter phenomenon is also encountered when monitoring people, where inaccuracy in sensing the position within rooms may be rather substantive.

It needs to be noted that the position of the stay is induced from the trajectory itself; without using any coordinates of potential, predefined stay locations. This is natural for vehicles that may stop anywhere. Due to privacy constraints, when monitoring the position of people, it is often desirable not to label the potential stay locations or to compile any databases of such places [18].

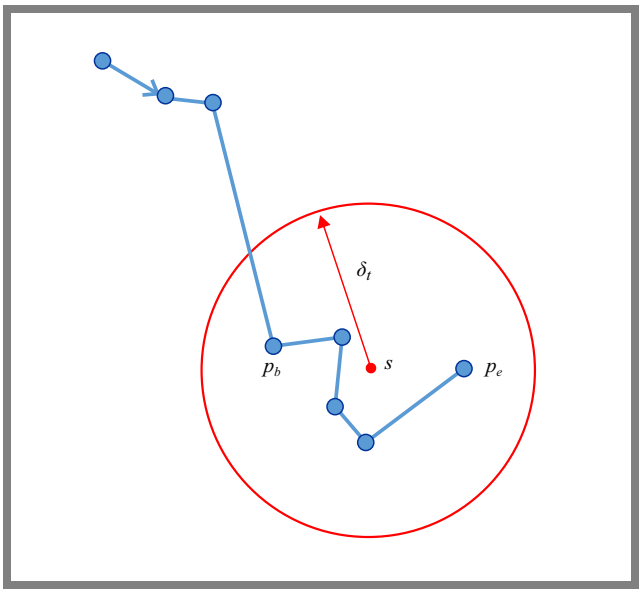

Fig. 1. Example of a stop (stay): $t_e - t_b > \delta_t$ must hold.

A variant of the above problem is possible, in which the stop condition – Eq. (1) – is modified by replacing demand $s \in \mathbb{R}^2$ with the requirement stating that s needs to be one of points p_i on the trajectory fragment recorded so far. This slight modification would be motivated by practical implications. When we show information concerning the stay to the user, it is better to show a specific and real GNSS probe and identify it as the place at which the vehicle is present, instead of presenting a hypothetical and artificial position, much less likely to be convincing for the user. Also, the process of designing simple algorithms for such a variant may be easier. However, this modification is not used later. Optionally, we could require that the algorithm:

- outputs a stable position (stay point) s,
- outputs an input element index b at which the stay commenced,
- outputs "Start" while processing the first input element d_i such that the fragment of input data from any index $b < i$ to i does not fulfill the condition (1).

Outputting a stable position may be necessary for the system to present the results to the user or to process this information further, in accordance with specific information. The same applies to the stay commencement index b. Outputting "Start" informs that the stay period is ending and the object is moving from the stay point. Then, the algorithm may be reset (reinitialized) and may look for another stop in the suitably renumbered remainder of the input stream.

Note that the stable position s is not necessarily uniquely defined for a given input sequence d_i. There may exist many s-es fulfilling Eq. (1). If we require s to be returned by the algorithm, we shall be satisfied with any of the candidate values.

The same applies to index b, describing the beginning of the stay period. It may happen that if a trajectory fragment delimited by input sequence indexes b and e fulfills the stop condition given by Eq. (1), then some shorter fragment delimited by $b' \geqslant b$ and e also fulfills the stop condition. This will happen whenever $t_e - t_{b'} > \delta_t$. Thus, whenever we require that a pair of stay fragment bounds (b, e) be returned by the algorithm, we shall be satisfied with any pair meeting the stop condition.

Note that in problem $P(\delta_p, \delta_t)$, we require a "Stop" to be output at iteration e at the latest. This actually means that Stop should be output as soon as the stay may be proved based on the so-far revealed fragment of the input sequence $(d_1, \ldots, d_j)$.

Definition of the problem does not determine how many times a "Stop"' may be output for a given stay. This is an intentional technical omission aiming to facilitate the further analysis. To avoid "hiccups", only the first "Stop" may be counted, and the subsequent occurrences of "Stop" may be ignored. The algorithm proposed later does not suffer from any hiccups.

Although the proposed algorithm is equipped with some additional features described above, i.e., elimination of hiccups or an additional output, we will not deal with these extensions in its formal analysis.

6. Solving Algorithm

Recall that in [15], a problem similar to the proposed problem $P(\delta_p, \delta_t)$ is considered, but in a non-stream setting. The algorithm receives the entire set $[d_1, d_2, \ldots, d_m]$ as input data. The stop condition is formulated similarly to Eq. (1) and is checked directly in the algorithm. This results in a double loop where both the inner and the outer loops iterate over the input data. The memory complexity of the algorithm in the GNSS stream length is $\mathcal{O}(m^2)$ and the time complexity is $\mathcal{O}(m^2)$. With the trajectory length of 1000 probes and in a system monitoring hundreds of thousands of objects, this could be too wasteful. Instead, a much more effective stream algorithm is proposed, at the cost of some inaccuracy. The description of the algorithm is divided into two parts: initialization and consumption of an input data element d_i.

The idea behind the proposition is as follows. The algorithm is supposed to detect stays without any prior knowledge of potential stay locations (rooms, flats, shops, etc.). It should infer stay locations from the trajectory. However, if the potential stay location was known in advance, it would be easy to detect stopping at this place by relying on stream processing. Only the lowest time value should be remembered, such that between that value and now, the trajectory did not leave the sphere around the potential stay location, with its radius equaling δ_p. In an iteration in which this time value becomes distant from the current moment by more than δ_t, a Start is detected. Since the potential stay location is unknown a priori, this action needs to be repeated for all points in $\mathbb{R}^n$. This is certainly infeasible. Thus, the set of potential stay locations is first reduced to a regular grid in $\mathbb{R}^2$. This grid still is infinite, but we can easily reduce it by considering, in the i-th iteration, only those grid points which are within the sphere of radius δ_p around p_i. Point outside this sphere cannot be stay locations, since p_i is too distant from them.

Let us precisely define the grid of points.

Definition 1. The ε-grid $G_x i\varepsilon \subset \mathbb{R}^2$ is equivalent to set $\{(i\varepsilon, j\varepsilon) : i \in \mathbb{Z}, j \in \mathbb{Z}\}$.

Now, we can define an algorithm for problem $P(\delta_p, \delta_t)$. The algorithm is parametrized by a natural number $\varepsilon \in (0, 1)$, representing the guarantied precision of the solution. The main variables are:

- T – a mapping from $G_{\varepsilon \cdot \sqrt{2}}$ into $\mathbb{R}$. The algorithm will not try to give an argument that is not stored in the mapping. An empty mapping means that no correspondence is stored in the mapping. T is a typical key-value map compiled using a popular programming language. $T(p)$ will be t_i for "the oldest possible stay commencement index b" for a given candidate p for the steady position.
- B – a similar mapping from $G_{\varepsilon \cdot \sqrt{2}}$ into $\mathbb{N}$. $B(p)$ will be the "the oldest possible stay commencement index b" for a given candidate p for the steady position. T similarly to the time we would have to remember in case of the a-priori known potential stay place.
- `stop_printed` – a Boolean variable saying whether there has already been some output produced by the algorithm and the last message has been "Stop", with additional information.

Algorithm 2 Provable ε-algorithm proposition – initialization.

1: Initialize T and B to empty mapping
2: `stop_printed`:=**false**

The purpose of the algorithm is as follows. When Algorithm 2 consumes the e-th element d_i of the input, only points from G are considered as candidates for the stay position. G is an approximation of the sphere with the center in p_i and radius δ_p. No point from outside of this sphere could be a proper s for a stay that becomes detectable precisely in this consumption, since it would be more distant from p_i than by δ_p, and this would violate the stop condition Eq. (1).

If some point $g^\star \in G$ belonged to similar spheres around previous p_i-s from the previous consumptions, then time $T(g^\star)$ is lower than $t_i - \delta_t$. Then again, when moving back in time until $T(g^\star)$, $g^\star$ has always been located not further than δ_p from a particular p_i. Hence, we can detect a stop in $s = g^\star$.

This algorithm is practical in that it does not display any hiccups behavior, Stop and Start outputs are printed alternately and delimit the consecutive stay periods. Also, the algorithm does not need to be reinitialized after the first detection of a stop, as it detects consecutive stop periods by itself.

The following theorems describe the algorithm's accuracy.

Theorem 1. If for a problem $P(\delta_t, \delta_p)$, ε-algorithm 2-3 outputs a "Stop", then a sound and exact algorithm for this problem also returns a "Stop", at the e-th consumption at the latest.

Proof. Since in step 19 in the e-th consumption $T(g^\star)$ was less than $t_i - \delta_t$, step 9 could have been run in consumptions b to e. Hence, in this period $g^\star$ has always been within the appropriate G-s, the stop condition Eq. (1) is fulfilled with e. □

In other words, a recognition of Stop by the proposed algorithm is correct, though maybe delayed. This is not surprising. Perhaps there existed some older fragments of the trajectory during Stop, not deviating from the actual steady point by no more than δ_t, and thus the detection of Stop could happen earlier, but the algorithm that uses only an approximation of the set of potential steady points could overlook this fragment. The lag, however, does not have to be very bothersome in practice. It can be easily shown that the delayed detection of Stop does not overlap with Start immediately following the moment at which a precise algorithm would detect Stop. Therefore, the delay works within the period of the stay.
Now, sensitivity of the algorithm will be assessed.
Theorem 2. Assume a sound algorithm for problem $P(\bar{\delta}_t, \bar{\delta}_p)$ with input stream (d_i) returns Stop in the e-th consumption of input data. Then proposed ε-algorithm for problem $P(\bar{\delta}_t, \bar{\delta}_p + \varepsilon)$ with the same input stream d_i also returns Stop in the e-th consumption at the latest. In other words, we can achieve 100% sensitivity of the algorithm by slightly decreasing δ_p to which the algorithm is tuned.

Proof. Since the sound algorithm detects Stop in consumption e, the stop condition Eq. (1) is fulfilled with $s = s^\star$ for e. It is easy to verify that there exists $\bar{g}$ in $G_{\varepsilon\cdot\sqrt{2}}$ with $\mathrm{dist}(\bar{g}, s^\star) \leqslant \varepsilon$. By the triangle inequality, p_i-s in consumptions $i = b+1, \ldots, e$ were at the distance from $s^\star$ not greater than $\mathrm{dist}(p_i, \bar{g}) + \mathrm{dist}(\bar{g}, s^\star)$. Hence, using the stop condition $\mathrm{dist}(p_i, s^\star) \leqslant \bar{\delta}_p + \varepsilon$ in these consumptions. This means that $T(g^\star)$ could not be changed in consumptions $b+1, \ldots, e$ (step 9 was not run in these consumptions). The condition from step 15 was satisfied and Stop was output iteration e at the latest. □

Algorithm 3 Provable ε-algorithm proposition – algorithm's reaction to element $d_i = (t_i, p_i)$ in the input stream.

1: $G := \{p \in G_{\varepsilon\cdot\sqrt{2}} : \mathrm{dist}(p, p_i) \leqslant \delta_p\}$
2: Set T' to empty mapping
3: Set B' to empty mapping
4: **for** $g \in G$ **do**
5: **if** g within the set of keys currently stored in T **then**
6: Add the correspondence g – $T(g)$ to T'
7: Add the correspondence g – $B(g)$ to B'
8: **else**
9: Add the correspondence g – t_i to T'
10: Add the correspondence g – i to B'
11: **end if**
12: **end for**
13: $T := T'$
14: $B := B'$
15: **if** $\exists\, g \in G :\ t_i - T(g) > \delta_p$ **then**
16: **if not** `stop_printed` **then**
17: Output "Stop"
18: Set $g^\star$ to some g satisfying the expression under the above quantifier
19: Output "Stop"
20: Output "Stop position s="; Output $g^\star$
21: Output "Stop begin index b="; Output $B(g^\star)$
22: Output "Stop begin time ="; Output $T(g^\star)$
23: `stop_printed` := **true**
24: **end if**
25: **else**
26: **if** `stop_printed` **then**
27: Output "Start"
28: Output "Start iteration index="; Output i
29: Output "Start iteration time="; Output t_i
30: `stop_printed` := **false**
31: **end if**
32: **end if**

In order not to overlook stops when using the proposed algorithm, one could slightly increase parameter δ_p. In most cases, this is practically viable. Parameter δ_p often represents the maximum possible noise affecting the GNSS position of an object (e.g., a car) that is fully immutable. Assuming that maximum possible noise is quite arbitrary, a larger maximum noise value can be easily set. If δ_p represents a diameter of a building or a room, it is also an approximation, since buildings and rooms are not sphere-shaped. This means that we can once again increase this parameter slightly. This way, we can certainly make the algorithm slightly oversensitive, but as long as the stay locations are sufficiently remote from each other, this should not be problematic.
The possibility that the algorithm detects stops quicker that the proper exact algorithm is, in reality, not surprising. It is not a fault per se, but a consequence of the oversensitivity. The algorithm with an artificially increased δ_p cannot falsely move the beginning of the detected stay backwards. It could even detect a false stop before the correct one, if the fluctuations of the object's position oscillate slightly more and exceed the original δ_p. Some complexity issues concerning the ε-algorithm for problem $P(\bar{\delta}_t, \bar{\delta}_p)$ still remain to be investigated. Mappings T and B are typical of popular programming languages. If real keys of these maps were in doubt for accuracy reasons, they can be essentially replaced with integer indexes in an efficient implementation, since they are points of some regular grid. The maps can be realized efficiently, e.g. by using binary trees.
Let us assume that the memory requirement for a k-elements map is $\mathcal{O}\big(k\mathrm{polylog}(k)\big)$. The maximum size of G, a set of plane grid points within a sphere of radius $\bar{\delta}_p$, can be easily verified to be $\mathcal{O}\left(\left(\bar{\delta}_p/\varepsilon\right)^2\right)$. T, T', B, B' are never of a greater size than G. The remaining variables do not dominate the memory complexity, which is finally the $\mathcal{O}\left(\mathrm{polylog}\left(\bar{\delta}_p/\varepsilon\right)\cdot\left(\bar{\delta}_p/\varepsilon\right)^2\right)$. This does not depend on the length of the input stream for the fragment consumed so far, so the algorithm truly is a stream algorithm.
The initialization costs clearly $\mathcal{O}(1)$ time. Assume dereferencing the value in the map costs $\mathrm{polylog}(k)$, and adding a correspondence to the map costs $\mathrm{polylog}(k)$ where k is the maximum possible number of map elements. The complex-

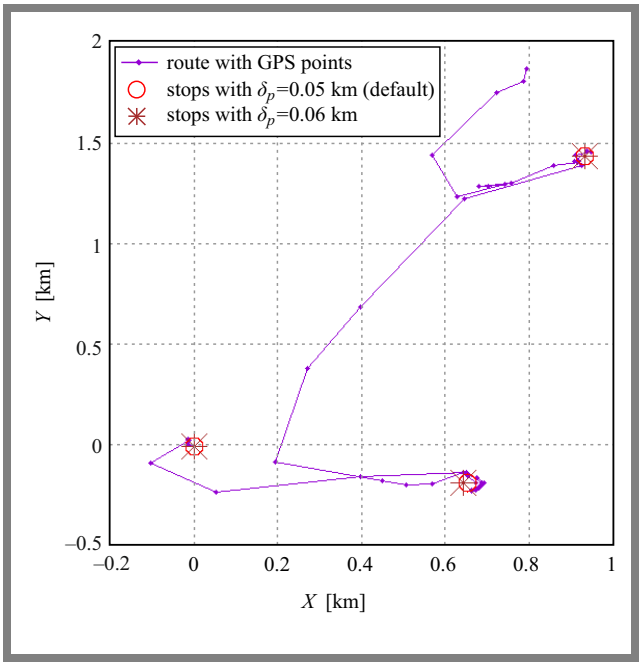

Fig. 2. Example route A, detected stay points for $\delta_p = 0.05$ and $\delta_p = 0.06$.

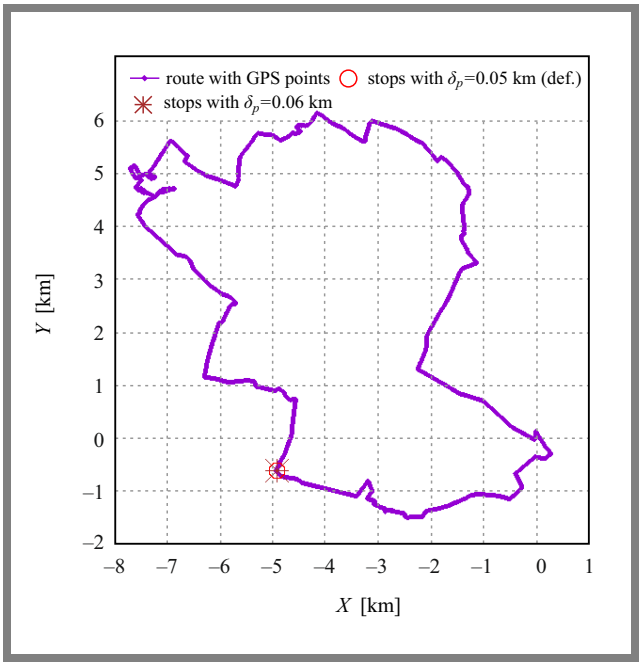

Fig. 3. Example route B, detected stay points for $\delta_p = 0.05$ and $\delta_p = 0.06$.

ity of one input element consumption is dominated by the $\mathcal{O}\left(\text{polylog}\left(\bar{\delta}_p/\varepsilon\right) \cdot \left(\bar{\delta}_p/\varepsilon\right)^2\right)$ additions of correspondences to maps. The total complexity of one input element consumption is finally $\mathcal{O}\left(\text{polylog}\left(\bar{\delta}_p/\varepsilon\right) \cdot \left(\bar{\delta}_p/\varepsilon\right)^2\right)$.
This is usually essentially less than the rank of the square of the input trajectory length, as typical for simple algorithms. However, accuracy ε should not be taken to be unnecessarily small, since it affects the problem's complexity through an inverse-square function.

7. Numerical Illustration

The operation of algorithm 2–3 will be demonstrated on several simple, real life GNSS traces, in order to validate the algorithm and to observe its sensitivity to changes of some basic parameters.
Three GPS route traces generated by volunteers on their journeys or walks were used in the experiments. They were obtained from the Open Street Map "traces" service, available under an Open Data Commons Open Database License [19]. Three up-to-date OSM examples of different lengths have been selected:

- Example A – OSM trace 11208248, 49 probes, approximately 15 h for the GPS device switched on, the time distance was varying between the ranks of one second and of one minute,
- Example B – OSM trace 11208345, 4620 probes, approximately 3 h, approximate time distance between consecutive probes 2 s,
- Example C – OSM trace 11208347, 6440 probes, approximately 7 h, approximate time distance between consecutive probes 2 s.

We used timestamps, longitudes, and latitudes from the traces. However, the proposed algorithm uses an Euclidean space to represent object positions. Hence, a transformation converting the longitude-latitude space into the $X - Y$ Euclidean plane was used, where the units are kilometers. This transformation was affine. Coordinate X was an affine function of longitude, coordinate Y was an affine function of latitude. The coefficients of the functions were selected so that the first probe of the trace yields coordinates $(0, 0)$, and that in the direct surroundings of this point, X and Y properly represent distance in km. This approximation around the first probe from the trace was used also for other probes – we ignored the Earth's curvature, since the routes were of the local variety. Various times were expressed throughout the experiments in seconds from the first trace frame's timestamp. Algorithm 2–3 was run using these routes. The default settings were as follows: $\delta_t = 300$ s, $\delta_p = 0.05$ km, $\varepsilon = 0.01$. We observed the output of the algorithm regarding the stays detected.
In the first phase, the stay radius parameter δ_p was varied (it was increased slightly from 0.05 to 0.06 km). We observed whether this increase, representing the surrogate increasing of δ_p required by the analysis of the algorithm, exerts a substantial impact on the detection of the stay.
In the second phase, grid pitch parameter ε was increased from the default value of 0.01 to 0.03 km. This was a considerable decrease of the algorithm's accuracy. The pitch of 0.03 was only slightly lower than the stay radius δ_p equaling 0.05. We wanted to see whether such a setting would substantially deteriorate detection efficiency. The routes and stop points s, stop commencement iteration indexes b and times t_b, as well as the start messages signaling the end of the stay generated by the algorithm for the different settings are presented in Figs. 2–7 and the details, in a slightly abbreviated format, are shown in Tab. 1.

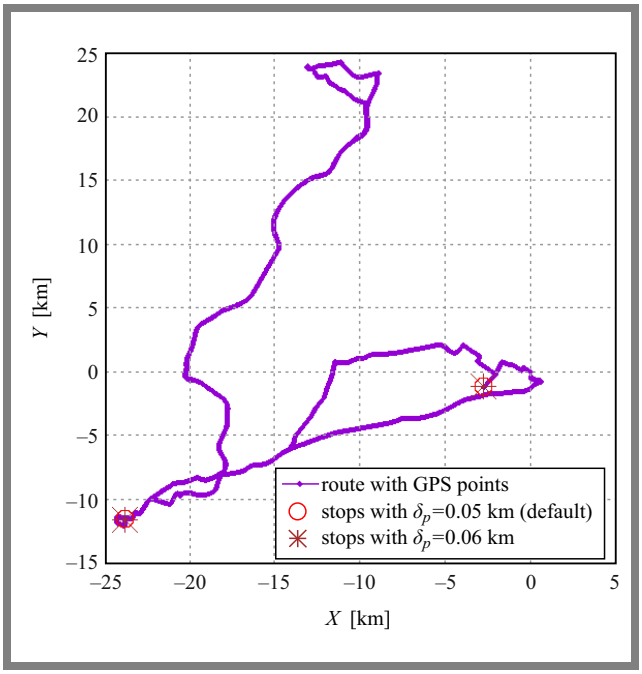

Fig. 4. Example route C, detected stay points for $\delta_p = 0.05$ and $\delta_p = 0.06$.

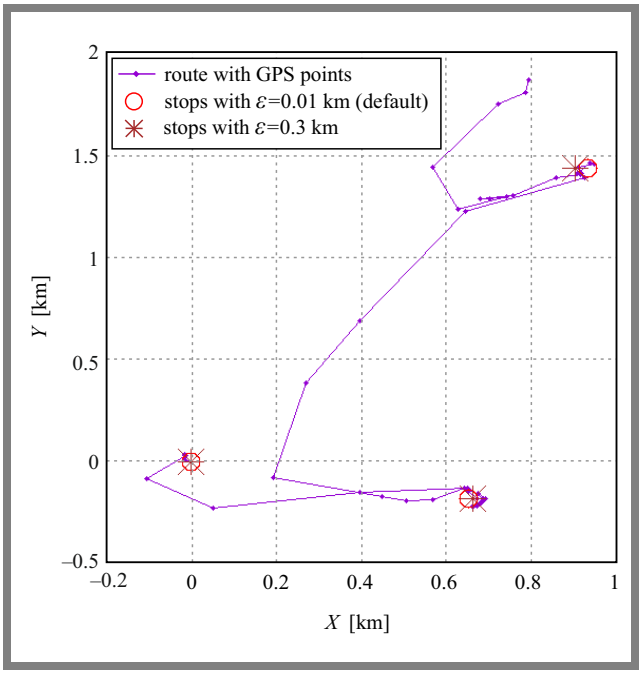

Fig. 5. Example route A, detected stay points for $\varepsilon = 0.01$ and $\varepsilon = 0.03$.

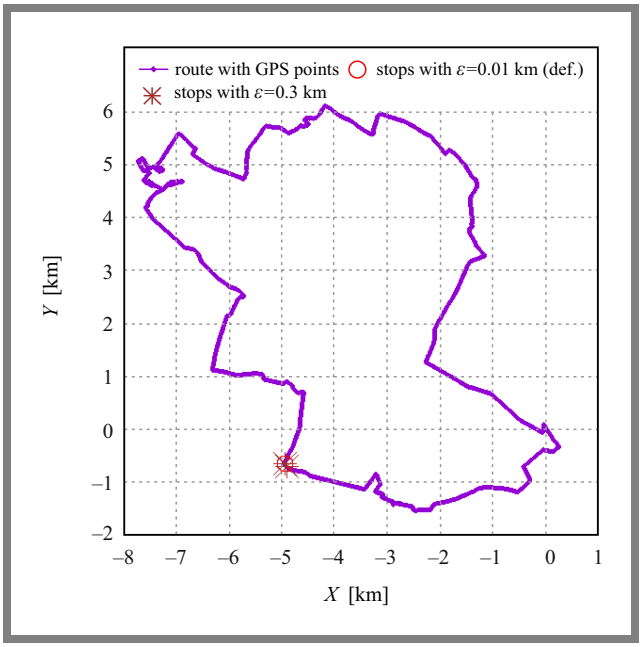

Fig. 6. Example route B, detected stay points for $\varepsilon = 0.01$ and $\varepsilon_p = 0.03$.

The experiment seems to have validated the algorithm. Stops are detected at reasonable locations: places of concentration of specific points – as visible in example A, around point $(0, 0)$, which is the start of the trajectory. We can suspect that the volunteer had switched the GPS device on a while before the journey actually commenced. Other detections appear to take place at locations in which the object clearly deviates away from the route.

The increase in stop radius δ_p did not impact the detection of particular stays. As foreseen in this analysis, an increase in δ_p may lead to slightly earlier stop detections. Also, the stay point s was occasionally moved by a distance in the rank of 10 m. This should not raise the level of anxiety, since the stop point does not need to be unique and the displacement was considerably smaller than the stay radius δ_p of 60 m. Insignificant reaction of the algorithm to the increase in δ_p is satisfactory, since such an increase is necessary to counteract a slight inaccuracy of the algorithm.

The increase in the grid pitch parameter ε from 0.01 m to 0.03 m also resulted in insignificant consequences to the place and time of detected stops. One additional detection occurred in such a scenario. The time complexity is over-quadratic in the ratio of δ_p/ε. For ε = 0.03, this ratio is as low as 5/3, and the algorithm still exhibits a fair behavior. While taking large ε values, the surrogate increasing δ_p to a sufficient level should be applied simultaneously. Although we have not done so, the algorithm was still able to work in a satisfactory manner.

8. Conclusions and Further Work

We have shown that the problem of stop detection in monitoring moving objects, when defined crisply and in the stream regime, is scientifically important, complex, and should not be overlooked. Many nuances need to be taken into consideration. Time limits of a stay are ambiguous: a larger time segment within which the trajectory points are close to the stay point s might be as good as some time segment contained in that, provided that both the segment lengths exceed the definitional time threshold δ_t.

However, with a stream regime we might not see the data from whole outer interval as early as from the inner one. A question arises which segment to report, and when. We have actually looked for a single stop in this paper, but if when we

require our algorithm to report all stops, the time limits ambiguity of stops probably will cause further problems with possible overlapping of stops and ambiguity of the number of stops. It must be stressed that our algorithm processes the multi-stop case in a reasonable manner, however, this behavior is not investigated formally.

We showed how the problem can be solved effectively, i.e, within the stream processing regime. Reaching this level an effectiveness might be crucial in large-scale applications. We gave an algorithm, which is inaccurate, but of provable limits of its inaccuracy. This algorithm seems to work soundly in the first tests.

Many directions of further research come to mind. A more precise definition of the stop problem is desired, that accounts for more nuances. Actually, describing the abovementioned nuances and ambiguities could require quite a mini-theory.

A more thorough analysis of the complexity of the problem would be welcome. First of all, a lower bound on the effectiveness of the solving algorithms should be given. It seems possible that the considered problem cannot be solved exactly with a stream algorithm, similarly to many other mathematical problems. For this sake, a counterexample could be given. Perhaps some tradeoff line matching the solution inexactness with time complexity of an iteration of stream algorithm could be given.

It seems appealing to define the θ-approximation of the solving algorithms, where the real number θ says how we should artificially change δ_p for the algorithm in order to attain the hundred percent od sensitivity. This is a formalization of what we did in Theorem 2. This tool could be used as the measure of inexactness of algorithms.

The sensitivity of the problem to changes in its parameters could also be investigated. For example, the detected stays and their time bounds should not change too rapidly with a change of δ_p. For this sake, the problem definition could be modified. Alternatively, some properties of the trajectory itself, naturally expected in practice, could be formulated that prevent such a behaviour.

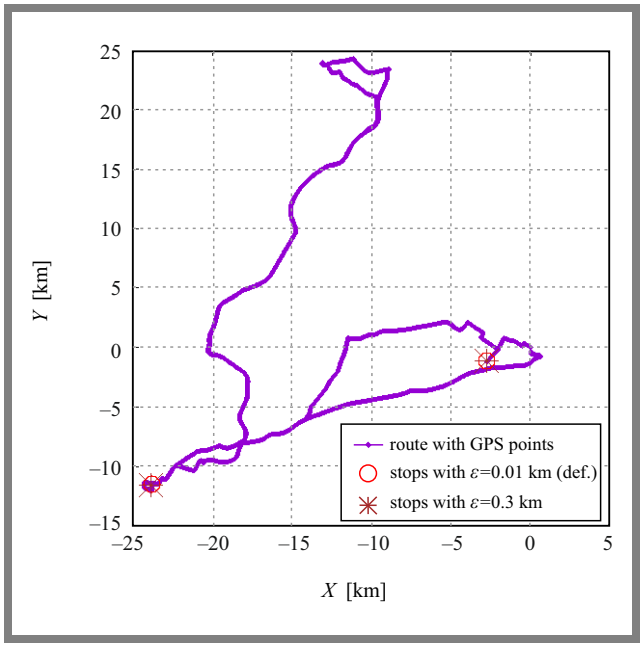

Fig. 7. Example route C, detected stay points for $\varepsilon = 0.01$ and $\varepsilon = 0.03$.

Most importantly, new solving algorithms should be searched for. In particular, creating grid points in our proposed algorithm is a bit complex and results in somewhat increased time of the iteration of our stream algorithm. Perhaps selecting all possible points from the intersection of the ball and the grid might be avoided. Our algorithm iteration (the consumption of the new data batch) costs $\mathcal{O}\left(1/(\theta-1)^2\right)$ time. Perhaps only one characteristic point can be taken instead of all points in the intersection that could construct a much faster algorithm with a reasonable proximity of the solution, say θ equal to 1.5 or 2. As discussed in Section 6, such inaccuracy might be still useful, e.g., when δ_p describes the noise level of GNSS, which is not very precisely known. Also, our attention should be directed to randomized algorithms, i.e., algorithms that internally generate sequences of randomly-chosen numbers and use them in the solving process [17].

Such algorithms are known to have lower complexity in many cases, at the inessential cost of yielding the solution only with a high probability (that can be arbitrarily close to 1), not for certain. Quite possibly, this would help overcome the inverse-quadratic dependency of the inexactness of solution, that holds for our proposition.

References

[1] A. Domin, D. Spruijt-Metz, D. Theisen, Y. Ouzzahra, and C. Vogele, "Smartphone-based Interventions for Physical Activity Promotion: Scoping Review of the Evidence over the Last 10 Years", *JMIR mHealth uHealth*, vol. 9, no. 7, pp. 638–648, 2021 (https://doi.org/10.2196/24308).

[2] Y. Ye, Y. Zheng, Y. Chen, J. Feng, and X. Xie, "Mining Individual Life Pattern Based on Location History", in: *Tenth International Conference on Mobile Data Management: Systems, Services and Middleware*, Taipei, Taiwan, 2009 (https://doi.org/10.1109/MDM.2009.11).

[3] S.E. Wiehe *et al.,* "Using GPS-enabled Cell Phones to Track the Travel Patterns of Adolescents", *International Journal of Health Geographics*, vol. 7, art. no. 22, 2008 (https://doi.org/10.1186/1476-072X-7-22).

[4] S.S. Dukare, D.A. Patil, and K. Rane, "Vehicle Tracking, Monitoring and Alerting Systems: A Review", *International Journal of Computer Applications*, vol. 119, no. 10, pp. 39–43, 2015 (https://doi.org/10.5120/21107-3835).

[5] Traccar: GPS Tracking Software - Free and Open Source System [Online]. Available: https://www.traccar.org.

[6] P. Deshmukh, A. Bhajibhakre, S. Gambhire, A. Channe, and N. Deshpande, "Survey of Geofencing Algorithms", *International Journal of Computer Science Engineering Techniques*, vol. 3 no. 2, 2018 (http://www.ijcsejournal.org/volume3/issue2/IJCSE-V3I2P1.pdf).

[7] Y. Ge, H. Xiong, C. Liu, and Z. Zhou, "A Taxi Driving Fraud Detection System", in: *2011 IEEE 11th International Conference on Data Mining*, Vancouver, Canada, pp. 181–190, 2011 (https://doi.org/10.1109/ICDM.2011.18).

[8] J.B. Oliva, "Anomaly Detection and Modeling of Trajectories", *M.Sc. Thesis*, Carnegie Mellon University, Pittsburgh, USA, 2012 (https://apps.dtic.mil/sti/citations/ADA566110).

[9] Y. Bu, L. Chen, A.W. Fu, and D. Liu, "Efficient Anomaly Monitoring over Moving Object Trajectory Streams" in: *Proc. of 15th ACM*

Tab. 1. Output of the algorithm for varying δ_p and ε.

Problem A
Default ($\delta_p = 0.05, \varepsilon = 0.01, \delta_t = 300$)

`STOP i=3, s=(0.00,0.00), b=2, tb=0.00`
`START i=7`
`STOP i=16, s=(0.65,-0.18), b=11, tb=14460.00`
`START i=27`
`STOP i=38, s=(0.93,1.44), b=34, tb=16024.00`
`START i=41`

$\delta_p = 0.06$

`STOP i=3, s=(0.00,0.00), b=2, tb=0.00`
`START i=7`
`STOP i=15, s=(0.64,-0.18), b=10, tb=14400.00`
`START i=27`
`STOP i=38, s=(0.93,1.44), b=34, tb=16024.00`
`START i=42`

$\varepsilon = 0.03$

`STOP i=3, s=(0.00,0.00), b=2, tb=0.00`
`START i=7`
`STOP i=15, s=(0.66,-0.18), b=10, tb=14400.00`
`START i=27`
`STOP i=38, s=(0.90,1.44), b=34, tb=16024.00`
`START i=41`

Problem B
Default ($\delta_p = 0.05, \varepsilon = 0.01, \delta_t = 300$)
`STOP i=3289, s=(-4.91,-0.62), b=3138, tb=6297.00` `START i=3305`
$\delta_p = 0.06$
`STOP i=3289, s=(-4.91,-0.62), b=3137, tb=6297.00` `START i=3311`
$\varepsilon = 0.03$

`STOP i=3289, s=(-4.92,-0.63), b=3138, tb=6297.00`
`START i=3307`
`STOP i=3309, s=(-4.89,-0.69), b=3156, tb=6301.00`
`START i=3311`

Problem C
Default ($\delta_p = 0.05, \varepsilon = 0.01, \delta_t = 300$)

`START i=395`
`STOP i=2023, s=(-23.78,-11.48), b=1870, tb=3722.00`
`START i=2200`

$\delta_p = 0.06$

`STOP i=330, s=(-2.73,-1.12), b=177, tb=359.00`
`START i=396`
`STOP i=2018, s=(-23.81,-11.52), b=1867, tb=3721.00`
`START i=2203`

$\varepsilon = 0.03$

`STOP i=330, s=(-2.73,-1.08), b=177, tb=359.00`
`START i=396`
`STOP i=2018, s=(-23.82,-11.52), b=1867, tb=3721.00`
`START i=2201`

SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, 2009 (https://doi.org/10.1145/1557019.1557043).

[10] L.H. Tran, Q.V.H. Nguyen, N.H. Do, and Y. Zhixian, "Robust and Hierarchical Stop Discovery in Sparse and Diverse Trajectories", *Infoscience. EPFL. Technical Reports*, 2011, https://infoscience.epfl.ch/record/175473.

[11] L. Killars, B. Schouten, and O. Mussmann, "Stop and Go Detection in GPS-position Data – Discussion Paper", *CBS. Discussion Paper*, Netherlands, 2020 https://www.researchgate.net/publication/338402776_Stop_and_Go_detection_in_GPS-position_data_-_Discussion_Paper.

[12] G. Cich, L. Knapen, T. Bellemans, D. Janssens, and G. Wets, "TRIP/STOP Detection in GPS Traces to Feed Prompted Recall Survey", *Procedia Computer Science*, vol. 52, pp. 262–269, 2015 (https://doi.org/10.1016/j.procs.2015.05.074).

[13] H. Safi, B. Asemi, M. Mesbah, and L. Ferreira, "A Trip Detection Method for Smartphone-assisted Travel Data Collection", in: *Proceedings of the Transportation Research Board (TRB) 95th Annual Meeting*, Washington, USA, pp. 1–18, 2016 https://eprints.qut.edu.au/125568/.

[14] L. Gong, T. Yamamoto, and T. Morikawa, "Identification of Activity Stop Locations in GPS Trajectories by DBSCAN-TE Method Combined with Support Vector Machines", *Transportation Research Procedia*, vol. 32, no. 3, pp. 146–154, 2018 (https://doi.org/10.1016/j.trpro.2018.10.028).

[15] R. Montoliu and D. Gatica-Perez, "Discovering Human Places of Interest from Multimodal Mobile Phone Data", in: *Proc. of the 9th International Conference on Mobile and Ubiquitous Multimedia*, Limassol, Cyprus, 2010 (https://doi.org/10.1145/1899475.1899487).

[16] J.H. Kang, H. Jong, W. Welbourne, B. Stewart, and G. Borriello, "Extracting Places from Traces of Locations", in: *Proc. of 2nd ACM Intl Workshop on Wireless Mobile Applications and Services on WLAN Hotspots*, Philadelphia, USA, pp. 110–118, 2004 (https://doi.org/10.1145/1024733.1024748).

[17] C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd Edition, MIT Press, London, UK, 2001 (ISBN: 9780262032933).

[18] T.M.T. Do and D. Gatica-Perez, "The Places of Our Lives: Visiting Patterns and Automatic Labeling from Longitudinal Smartphone Data", *IEEE Transactions on Mobile Computing*, vol. 13, no. 3, pp. 638–648, 2014 (https://doi.org/10.1109/TMC.2013.19).

[19] The Open Street Map service, [Online]. Available: https://www.openstreetmap.

Paweł M. Białoń, Ph.D.
Advanced Information Technologies Department
https://orcid.org/0000-0002-7781-9212
E-mail: P.Bialon@il-pib.pl
National Institute of Telecommunications, Warsaw, Poland
https://www.gov.pl/web/instytut-lacznosci