Noname manuscript No.
(will be inserted by the editor)

PAS3-HSID: A Dynamic Bio-Inspired Approach for Real-time Hot Spot Identification in Data Streams

Rebecca Tickle · Isaac Triguero · Grazziela P. Figueredo · Mohammad Mesgarpour · Robert I. John

Received: date / Accepted: date

Abstract Introduction: Hot spot identification is a very relevant problem in a wide variety of areas such as health care, energy or transportation. A hot spot is defined as a region of high likelihood of occurrence of a particular event. To identify hot spots, location data for those events is required, which is typically collected by telematics devices. These sensors are constantly gathering information, generating very large volumes of data. Current state-of-the-art solutions are capable of identifying hot spots from big static batches of data by means of variations of clustering or instance selection techniques that pre-process the original input data, providing the most relevant locations. However, these approaches neglect to address changes in hot spots over time.

Method: This paper presents a dynamic bio-inspired approach to detect hot spots in big data streams. This computational intelligence method is designed and applied to the transportation sector as a case study to identify incidents in the roads caused by heavy goods vehicles. We adapt an immune-based algorithm to account for the temporary aspect of hot spots inspired by the idea of pheromones, which is then subsequently implemented using Apache Spark Streaming.

Results: Experimental results on real datasets with up to 4.5 million data points - provided by a telematics company - show that the algorithm is capable of quickly processing large streaming batches of data, as well as successfully adapting over time to detect hot spots.

R. Tickle, I. Triguero, and Robert I. John
The Automated Scheduling Optimisation and Planning Research Group
School of Computer Science, University of Nottingham, NG8 1BB, United Kingdom
Tel.: +441158466416
E-mail: Isaac.Triguero@nottingham.ac.uk

G.P. Figueredo
The Advanced Data Analysis Centre
School of Computer Science, University of Nottingham, NG8 1BB, United Kingdom

M. Mesgarpour
Microlise, Farrington Way, Eastwood, Nottingham NG16 3AG, United Kingdom

Conclusions: The outcome of this method is two-fold both reducing data storage requirements and demonstrating resilience to sudden changes in the input data (concept drift).

Keywords Hot Spots · Road Incidents · Instance Selection · Telematics Data · Big Data Streams · Computational Intelligence

1 Introduction

Hot spot identification (HSID) problems are present across several domains, such as health care, security, maintenance, energy or transport [4, 6, 11, 25]. A hot spot can be defined as a particular area with a high likelihood of occurrence of a certain event. Several HSID application opportunities are identifiable. In public health care, for instance, algorithms to determine hot spots could be employed for early detection of locations of an epidemic outbreak. In security, the government and population benefit from knowing specific areas of elevated crime rate. HSID methods can also be applied commercially, for example, by using mobile phone data to determine most frequently visited places and provide targeted marketing interventions. While these examples mostly belong to unrelated disciplines, their commonality is that the establishment of a set of hot spots relies on location data. Although the current widespread use of mobile devices, sensors and trackers facilitates data gathering, challenges regarding data retrieval, fusion and interpretation for HSID arise.

In this work we are interested in tackling the problem of processing and interpreting huge influxes of vehicle telematics data for HSID within the intelligent transportation systems (ITSs) context [27]. Research in ITSs aims to create methods, processes and devices to allow for improvements in driving performance as well as road economy and safety [24]. Logistics coupled with large transport networks has demanded the use of sensors and tracking devices (telematics) to achieve such goals. For vehicle incident HSID, telematics constantly records data on locations, date, time, direction, etc, ready to be exploited.

Traditionally, statistical methods have been employed to establish hot spots from historical data [6]. However, these methods may not be suitable for handling big amounts of data [31]. Data mining techniques have also been used to address this problem [1]. For example, clustering algorithms such as K-means [2] can group incidents based on distance, with each resulting cluster representing a hot spot. However, those clusters may not produce valid hot spots and do not provide information about their relevance. More recently, instance selection techniques [17], originally devised for data pre-processing [18] in classification tasks, have successfully been used to address the hot spot problem.

In [15], a computational intelligence technique based on immune systems [30], namely SeleSup HSID, was proposed to tackle HSID, addressing the main issues found with traditional approaches. This method adapted an immune-inspired instance selection algorithm [13] to detect vehicle incident hot spots and highlight their importance by means of a fitness value. Recently, [31] re-conceptualised the SeleSup HSID algorithm as a series of MapReduce-like operations [7] under the Apache Spark platform [33] to improve the efficiency of the method when dealing with huge volumes of data.

Despite its efficiency, this type of approach does not cope well with a constant influx of data that may vary over time, being unable to provide a timely answer and account for (sudden) changes in the distribution of the data (e.g. due to weather, new signalling or works in the road) when measuring the importance of the identified hot spots. The large data streams provided by vehicle telematics present new challenges [16, 21], as they produce an unbounded and 'potentially infinite' amount of data that it may not be feasible to store and process as one batch, resulting in a need for online processing [8, 23]. The use of data pre-processing techniques would help reduce the amount of data; however, current approaches do not deal effectively with the non-stationary characteristics of data streams as discussed in [28], and the SeleSup HSID algorithm suffers from the same issue.

The aim of this paper is to redesign the SeleSup HSID algorithm to tackle huge volumes of streaming data for vehicle HSID. We propose an adaptive SeleSup HSID algorithm that is inspired by a pheromone-based approach [9] to dynamically determine the importance of hot spots based on current and past data, eliminating old hot spots, and adding new relevant locations. The algorithm is designed under Apache Spark Streaming [34] as a number of MapReduce operations to parallelise the most time consuming operations of SeleSup, enabling the detection of hot spots in big data streams. We denote this method as PAS3-HSID (Pheromone-based Adaptive SeleSup Streaming algorithm for Hot Spot Identification). Developing a dynamic HSID technique motivates the global purpose of this work, which can be split into three main objectives:

- To design a hot spot detection technique based on pheromones that is capable of dealing with a time-varying scenario and potential concept drift on the stream of data.
- To reduce the size of telematics data that is stored by discarding irrelevant data and keeping representative hot spots together with their current relevance [5].
- To analyse the scalability of the proposed scheme in big data streams of vehicle incidents.

To test the performance of our model, we will conduct a series of experiments on big datasets of heavy goods vehicle incidents provided by Microlise[1], a UK-based company that provides telematics solutions to help fleet operators to reduce their costs and environmental impacts. By applying the proposed PAS3-HSID algorithm to these datasets, containing millions of HGV incidents, we will investigate the effect of different time windows, parameters and scalability capabilities. We also compare our method with the existing SeleSup HSID approach, identifying the benefits that our pheromone-based mechanism provides.

The remainder of this paper is organised as follows. Section 2 describes the background of HSID, instance selection and big data technologies. Section 3 presents the PAS3-HSID algorithm and its main characteristics. Section 4 discusses the experimental framework and presents the analysis of results. Finally, in Section 5 we summarise our conclusions.

2 Background

This section presents all the background information necessary to understand the remainder of this paper. Subsection 2.1 defines the hot spot identification problem

[1] https://www.microlise.com/

for the case of transportation and describes current approaches for batch data based on clustering and instance selection. Subsection 2.2 discusses the existing instance selection methods for data streams. Finally, Subsection 2.3 briefly introduces the big data technologies employed in this paper.

2.1 Hot spot identification in transportation

HSID consists of processing large amounts of location data for a particular problem. Because hot spots are established based on the proximity of event occurrences, a domain-specific distance measure should be defined. In some cases, this could simply be the physical distance between the locations of events; in others, additional constraints may be required when determining whether a specific event contributes to a hot spot or not.

One application of HSID is to transportation problems, and our specific case concerns heavy goods vehicle (HGV) incidents as the events of interest. These incidents indicate the driver's behaviour in some way; examples of such incidents are speeding, harsh braking and harsh cornering. Given a constant data stream of HGV incidents containing incident type, date, time and location, those areas of high likelihood of incident occurrence should be determined. The distance measure used is the distance between incidents, with the additional constraints requiring that incidents occur on the same road and have similar bearings.

HSID has to be accurate for all types of incidents at any location. In addition, it is desirable that the method identifies and reflects on the HSID process those changes in roads and driving behaviour that occur over time. Additionally, in scenarios such as those illustrated in Figure 1, several different indications of hot spots can be determined; however, not all of them provide satisfactory solutions for our problem, as discussed in Figueredo *et al.* [15]. For instance, those clusters indicated by blue circles (such as cluster A) represent good candidate solutions. Clusters B (with one instance, not considering neighbour incidents) and C (bigger ellipsis, where road direction is disregarded and multiple hot spot locations are included) represent invalid solutions. The solution to the problem posed should be able to provide only valid hot spots.

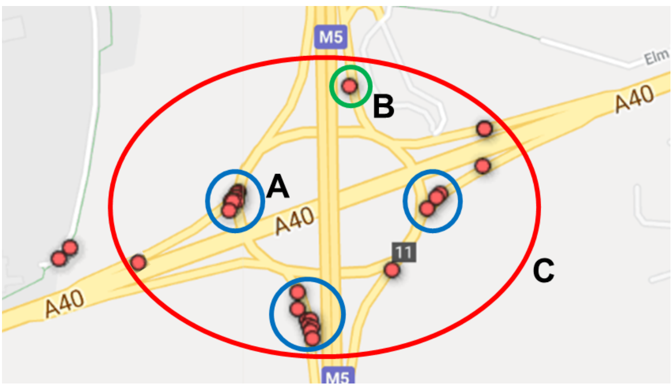


Fig. 1: Examples of possible hot spot clusters.

Statistical methods have often been used for hot spot identification. Three such methods are evaluated in [6], namely simple ranking, confidence intervals, and Empirical Bayes. These approaches all establish likely hot spots by comparing locations with sites that have similar characteristics. Simple ranking involves locations being ranked in descending order of crash frequency. The confidence interval technique determines that a site is unsafe if the observed number of crashes is greater than the average observed at similar sites. By taking into account both historical crashes at the location in question, and the expected number of crashes at comparable sites, Empirical Bayes performs the best of these three methods. However, these statistical approaches are not suitable for use with large volumes of data, and also rely on identification of comparable locations before hot spot identification can occur.

As discussed in Figueredo *et al* [15] the application of spatial clustering methods for this problem (such as density-based spatial clustering [12] and other techniques [19]) is ineffective. These techniques can require a predefinition of the number of clusters, which could reduce the accuracy of the hot spots obtained. Furthermore, they may produce elliptical clusters, such as that indicated by the red line (cluster C) in Figure 1, or require an adaptation for big data problems [29].

Recent work has employed instance selection techniques for the purpose of hot spot identification on large datasets. Instance selection [17] is a data preprocessing technique that is normally used to reduce the size of a dataset prior to it being used for data mining. This is achieved by removing data points that are redundant or noisy, leaving behind a smaller subset that is still representative of the original data, resulting in lower storage requirements and more efficient mining without compromising the accuracy of the results [20]. In the HSID context, the points remaining after instance selection are the hot spots.

An immune-inspired instance selection method, SeleSup [13,14,26], was successfully used in Figueredo *et al.* [15] to reveal hot spots. This method has an advantage over traditional clustering methods in that the number of 'cluster' centres is self-adaptive, and therefore no predefinition of the number of hot spots is required. However, the implementation of the algorithm shows reduced performance on datasets with millions of instances. The work done in Triguero *et al.* [31] aims to improve the performance of this algorithm by adapting it for implementation in Apache Spark. This implementation indicates the same hot spots for the datasets as the previous implementation, and also demonstrates an increase in performance for larger datasets, due to the distributed nature of the computation.

While the SeleSup method and its subsequent implementation in Spark performs well for large batch datasets, it is not suitable for HSID in a dynamic streaming environment. Our novel approach appropriately tackles the challenges of data streams, using instance selection as a technique. The next section discusses some of the existing instance selection methods for data streams in the literature.

2.2 Instance selection for data streams

Additional challenges become apparent when considering the application of instance selection to data streams, due to the dynamic nature of streams. The instances retained by the selection method must be representative of the current state of the stream and be able to update quickly as the distribution of the data changes over time (concept drift) [16]. As recently surveyed in [28], existing instance selection techniques do not cope well with the non-stationary characteristics of data streams. Here, we discuss some current approaches and consider whether they could be applied to the hot spot problem.

Klinkenberg [22] compares multiple methods for handling concept drift by selecting the number of instances to be used. These include an adaptive time window, batch selection, and weighting instances with respect to their age. The experiments showed that batch selection, where batches of data that seem to include a large number of outliers are eliminated, performed best, closely followed by the adaptive time window. Weighting instances gave the lowest performance, although was better than methods that did not adapt for concept drift. All of these methods use the assumption that the most recent examples are the most relevant, and do not account for recurring concept drift, where concepts that existed previously become relevant once more.

The instance-based learning on data streams (IBL-DS) algorithm proposed in [3] was developed to tackle the problem of concept drift for classification on data streams. This approach takes into account both the time that instances arrive, and the distance between instances to determine redundant or noisy points to remove. Older instances are also removed when the size of the case base will exceed a given maximum, whilst newer instances are safe from elimination to allow time to determine whether they are simply noise, or the beginnings of a new concept. For the scenario of hot spot identification, limiting the number of hot spots can have detrimental effects for the accuracy. In addition to this, IBL-DS results in the deletion of old instances even if they are still relevant to the current state of the data stream.

A different approach to instance selection for classification is to store only those instances that define the boundaries between classes, reducing the memory requirements of the model. One such example is presented in [35], where a data stream classification algorithm based on an artificial endocrine system is proposed. As the stream progresses, the maintained instances change, representing the evolving class boundaries. Although this mechanism works well for classification, it would not be suitable for hot spot identification, where there are no such boundaries to find.

In summary, existing instance selection techniques for data streams are not suitable for application to the hot spot identification problem. We require a method that, while adapting with respect to the most recently arrived instances, can also take into account previously established hot spots and incorporate them in the current set of hot spots in some way. It is also essential that the method does not rely on removing long-standing hot spots after a fixed time period, as these can be significant areas for HGV incidents. Instead, hot spots should be deleted based on an alternative measure of their importance.

2.3 Big data technologies

MapReduce [7] was developed by Google for the parallel processing of data across large clusters, and has a popular open-source implementation, Apache Hadoop. MapReduce computations are described in terms of two user-specified functions: *map* and *reduce*. These functions work on key/value pairs, defined based on the data to be processed. The map stage applies the given function to each input pair. The data is then shuffled so that all values for a particular key are grouped together, the result of which is then passed to the reduce function. This merges the values assigned to a key together, usually returning a single value per key. There are some cases for which Hadoop is not the most suitable choice, such as for iterative algorithms where data needs to be reused across computations, a task which it does not efficiently accomplish.

Other data processing frameworks exist that overcome these drawbacks. Apache Spark is one such example, introducing a distributed memory abstraction known as Resilient Distributed Datasets (RDDs) [33]. A Spark cluster consists of a driver node alongside multiple worker nodes, and RDDs allow data to be cached, or persisted, in main memory of these nodes, resulting in more efficient data reuse. The Spark programming interface provides several MapReduce-like operations that can be applied to RDDs, such as *map*, *reduce* and *filter*. There are also methods for moving data between nodes. These include *collect*, which fetches all elements of an RDD back to the driver node, and *broadcast*, which sends a read-only variable to all nodes.

Spark Streaming is an extension to Spark that treats data streams as a sequence of microbatches on which to perform computations [34]. It provides discretized streams (DStreams) as a programming model. DStreams are fundamentally a series of RDDs, with each RDD of the input DStream representing one batch, or interval. The programmer defines a sequence of operations to be applied to the incoming data, which Spark Streaming will apply as the data arrives. Intervals can be processed independently of each other, or alternatively window operations can be used to allow operations to be applied to multiple consecutive batches at once. Stateful transformations are also available and facilitate the sharing of data between intervals.

3 PAS3-HSID: Pheromone-based approach for adaptive HSID

Here we present our immune-inspired, pheromone-based adaptive SeleSup algorithm (PAS3-HSID) for hot spot identification in data streams. This algorithm is based on the existing SeleSup HSID method [15], with the additional consideration of how to establish a set of hot spots that can change over time in response to incidents arriving. We assume that the stream is split into time intervals, and that incidents arriving within one interval are allocated to one batch that is processed at the end of that interval.

The algorithm is designed with three main requirements in mind:

- Identification of hot spots from streamed incident data, taking into account the temporal nature of this data.
- Reduction of the volume of data that needs to be stored at each interval of the stream. Instead of storing all incidents that arrive per interval, the hot spots identified must represent a reduction in this data, resulting in lower storage requirements.
- Suitability for parallelisation, to enable an implementation that can efficiently compute hot spots for large batches. This is required because there is the potential for data to be arriving in very large batches due to the quantity being generated through HGV telematics, which would result in poor performance from a sequential implementation.

We first explain the algorithm from a general perspective in Subsection 3.1, before providing specific details of our Spark-based implementation, designed to process large batches of incident data in parallel, in Subsection 3.2.

3.1 PAS3-HSID details

The PAS3-HSID algorithm works by maintaining a state of current hot spots between time intervals of a data stream. At each interval, the algorithm receives as input a batch of new incidents I to be reduced. Using these incidents, as well as the hot spots from the previous interval, an updated set of hot spots is produced. Figure 2 shows how the state is repeatedly updated and fed into PAS3-HSID to determine future hot spots.

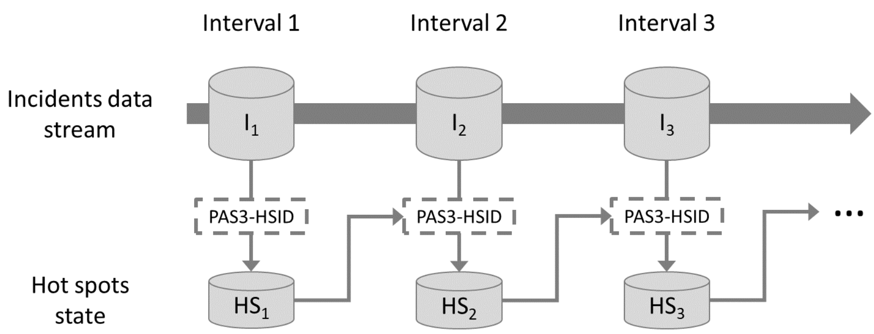


Fig. 2: Representation of how PAS3-HSID updates the hot spot state at each time interval of the data stream and then uses the state in the process of producing a new set of hot spots.

Each hot spot in the state is associated with a fitness value representing the strength of that hot spot. Higher fitness values indicate hot spots that have relatively recently gained a large number of incidents, whilst lower values represent hot spots with a smaller number of incidents, or those that have not been updated with new incidents for a while. A lower fitness value suggests that a hot spot is becoming less relevant to the current state of the data stream.

The fitness values $FV_1, FV_2, ..., FV_{\#HS}$ are initialised to the number of incidents included within the respective hot spot when it is first discovered, similar to how fitness values are decided in [15]. The state is updated at each interval through a pheromone-based mechanism that alters the fitness values accordingly. Any hot spots with a fitness value below a given threshold are discarded, ensuring that the set of hot spots remains representative of the current distribution of incidents. Using fitness values to determine when to remove hot spots ensures that they are not deleted based purely on how long they have existed for. Instead, we are also considering their relevance to the current state of the stream; in other words, whether a hot spot has recently had any incidents occurring in its vicinity.

Our use of pheromones is inspired by a similar mechanism utilised in ant colony optimisation (ACO) [9], a technique for finding short paths through graphs, based on the behaviour of ants in nature that deposit pheromones whilst finding food. In ACO, this idea is used to iteratively construct solutions to the shortest path problem, by getting a population of artificial ants to deposit pheromones on the edges of a graph.

The higher the pheromone value of an edge, the greater the probability of it being selected by ants at future iterations. Ants that generate good solutions will deposit larger amounts of pheromones than those that find worse solutions. In addition, an evaporation rate is also set, so that the pheromone values will decrease over time.

We can apply the pheromone idea to the fitness values of hot spots. Fitness values must be increased at each interval in relation to the number of incidents added to each hot spot, similar to depositing pheromones of the edges of the graph in ACO. Just as the edges that contribute to shorter paths receive more pheromone, hot spots that gain more incidents in a given interval will see their fitness value increase by a larger amount. We also require the fitness values to decrease over time, so that eventually hot spots will be removed after not gaining new incidents for some time. This ensures that the current set of hot spots is truly representative of the present state of the roads, and is equivalent to the evaporation of pheromones.

Algorithm 1: PAS3-HSID, a pheromone-based adaptive hot spot identification algorithm for data streams.

Require: HotSpots; Incidents; DecayRate; DeleteThreshold; MileageRange; PercentInitHotSpots

STAGE 1
if $HotSpots$.isEmpty **then**
  $HotSpots \leftarrow$ take $percentInitHotSpots \cdot |Incidents|$ from $Incidents$
  forall $HotSpots$ **do** $FV_h = 1$; $n_h = 0$;
  $Incidents \leftarrow Incidents - HotSpots$
else
  forall $HotSpots$ **do** $n_h = 0$;
end if
for all i in $Incidents$ **do**
  for all h in $HotSpots$ **do**
    $d \leftarrow$ calculate distance between h and i w.r.t distance measure
    if $d < MileageRange$ **then**
      $Incidents \leftarrow Incidents - i$
      n_h += 1
      break
    end if
  end for
end for

STAGE 2
forall $Incidents$ **do** $FV_i = 1$; $isCentre_i =$ false; $n = 0$;
for all i in $Incidents$ where $!isCentre_i$ **do**
  for all $j \mathrel{!=} i$ in $Incidents$ **do**
    $d \leftarrow$ calculate distance between i and j w.r.t distance measure
    if $d < MileageRange$ **then**
      $Incidents \leftarrow Incidents - i$
      n_j += 1; $isCentre_j =$ true;
      break
    end if
  end for
end for

STAGE 3
$newHotSpots \leftarrow HotSpots + Incidents$
for all h in $newHotSpots$ **do**
  $FV_h \leftarrow FV_h \cdot (1 - DecayRate) + n_h$
  if $FV_h < DeleteThreshold$ **then**
    $newHotSpots = newHotSpots - h$
  end if
end for
return $newHotSpots$ to be available at next interval

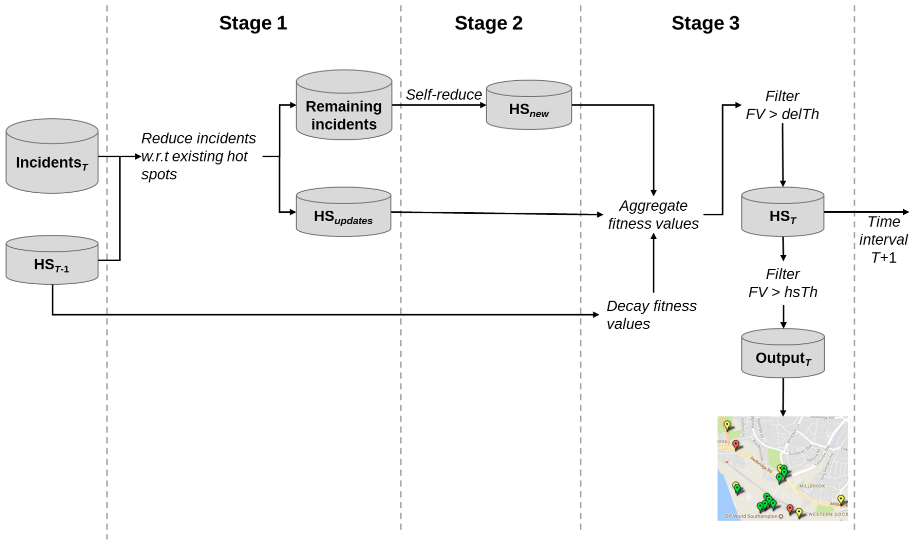
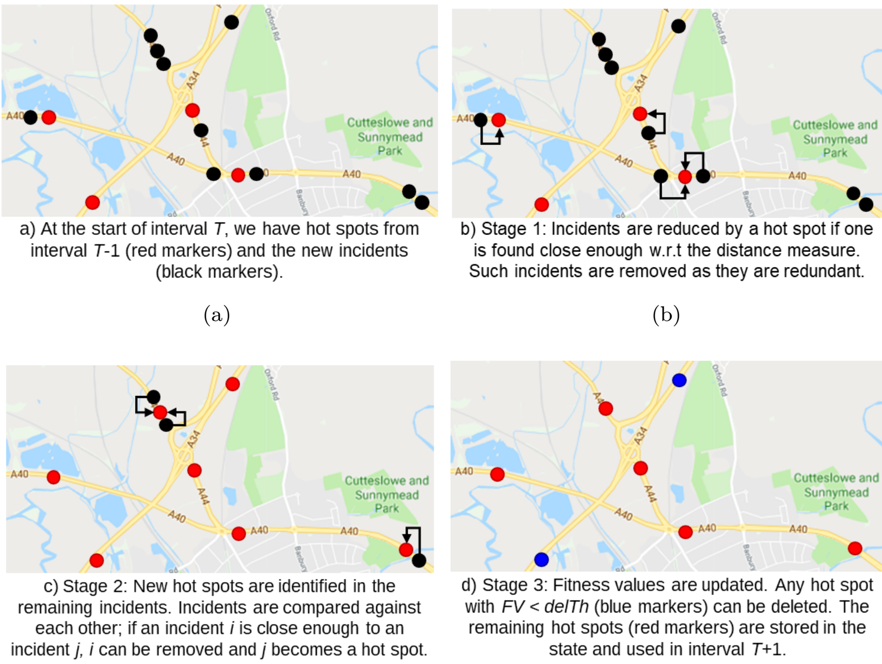


Fig. 3: Overview of the PAS3-HSID algorithm at a single time interval T. The hot spots that are output at the end of the interval can be visualised, processed or stored as required.

The algorithm consists of three main stages, as shown in Algorithm 1 and Figure 3, that take place at each interval of the stream. Figure 4 illustrates the process of determining current hot spots from a set of incidents and pre-existing hot spots.

1. The first stage of the streaming algorithm is based on Stage 2 of the original SeleSup HSID and involves using the existing hot spots HS to reduce the new batch of incidents I. This determines the incidents that can be discarded as their location in the road is already represented as a hot spot. Each incident i is compared with each hot spot h in turn, using a distance measure to decide how close i is to h. The distance measure used for vehicle incident HSID takes into account the incident location (latitude/longitude coordinates), bearing and address. Bearings must be within sixty degrees of each other, whilst the distance between locations is calculated as the Haversine distance [32]. If i is similar enough to any h, then i is said to be reduced by h; the presence of h in the hot spot set sufficiently represents the location of i, and therefore i is discarded as a redundant instance. Throughout Stage 1, we keep track of a value n_h for each h. This value is initialised to zero at the start of every interval, and is incremented each time h reduces an incident in the current batch. It is then used later in Stage 3 when recalculating the fitness value of h. Note that it is not necessary to ensure that an incident is reduced by the closest hot spot, as we are not aiming to find a precise location for the hot spot centre; rather, we want to find the general areas of the road where there are a high frequency of incidents. Therefore, an incident is reduced by the first hot spot found that it is close enough to, with respect to the distance measure. This has the additional advantage of being generally faster than finding the closest hot spot, which is important in the context of processing big data streams.



(a) (b)

(c) (d)

Fig. 4: Example of how PAS3-HSID computes hot spots at a time interval T.

There is also a special case of Stage 1, occurring at intervals when HS contains no hot spots. This is always the case in the first interval of the stream but may also happen at other points if there is a very low number of incidents for a prolonged period of time. In this situation an additional step is performed prior to the main part of Stage 1. This step replaces the empty HS with a small number of incidents randomly selected from I; these can then be used as if they were hot spot centres, to reduce the remainder of the incidents. This process is similar to that used at the start of the original method proposed in [15], where the recommendation is to use a low number of initial hot spots as it has no impact on the final number of hot spots and often results in a quicker runtime. Hence, we typically select 10% of I to be included in this set; however, this is a user-defined parameter and can be changed as desired. Any redundancies within these initial hot spots are removed, before Stage 1 proceeds as normal.

At the end of Stage 1, the incidents remaining in I are those that could not be reduced by any existing hot spots. These incidents are passed onto the next stage of the algorithm.

2. Stage 2 operates on those incidents that are left in the incident set I after Stage 1. These are incidents that could not be reduced by the existing hot spots, and therefore potentially represent new hot spot locations. The purpose of this second stage is to identify such new hot spots. The process used is similar to that used for the final step of the SeleSup HSID method. Each incident i is compared to

every other remaining incident, until an incident j is found that is close enough to reduce i, with respect to the distance measure. We then establish j as a new hot spot centre, and discard the redundant i. Note that once again, the aim is not to find the precise locations of hot spots, and therefore knowing that some i is within range of j is enough to declare j as a hot spot. For each incident j that becomes a new hot spot centre, a value n_j is incremented to indicate the number of incidents reduced by j. The result of Stage 2 is a set of new hot spot centres, representing road locations that have only recently had a high frequency of incident occurrence. These will be added to the hot spot state in the next stage.

3. The third and final stage performs the state update, using the information acquired from Stages 1 and 2 to produce a new hot spot state. Existing hot spots in the state have their fitness values recalculated using the pheromone-based mechanism. We define the following fitness value update formula, based upon the ACO pheromone update formula in [10]:

$$FV_h = FV_h \cdot (1 - dr) + n_h \tag{1}$$

First, each fitness value is decayed with respect to the decay rate dr, representing the decrease in relevance of hot spots over time. Then, the fitness values of those hot spots that reduced incidents in Stage 1, and are therefore still active, are increased by the value n_h (the number of incidents reduced by hot spot h). The new hot spots identified in Stage 2 are added to the state, with their fitness values initialised n_h. Finally, any hot spot with a fitness less than a specified deletion threshold $delTh$ is deemed to no longer be a hot spot, and is discarded. The resulting state will feed into the next stream interval to be used in the process of deciding the next set of hot spots.

Further filtering on the hot spot state can then be performed, to produce a subset containing those hot spots with a fitness value greater than a given hot spot threshold $hsTh$. This subset is then returned as the output hot spots of the algorithm at the current stream interval, to be stored and possibly used in further processing or visualisation. Hot spots with $delTh < FV < hsTh$ are not returned but are kept in the state and given a chance to develop a higher fitness value in the future.

3.2 Spark Streaming-based implementation

In this section we present the implementation details of PAS3-HSID in Apache Spark Streaming, parallelising most of the operations. We have chosen Spark Streaming as the big data framework with which to implement our algorithm, due to the algorithm's iterative nature; as previously stated, this is not well suited to Hadoop.

The implementation makes use of an RDD of key/value pairs to represent hot spots in the state, with each pair corresponding to a single hot spot. Hot spots are identified by a tuple $< lat, long >$ containing the latitude and longitude of the hot spot centre (the *key*). The *value* associated with a hot spot's key is any additional information about that hot spot required by the algorithm, such as its fitness value, bearing and address. The pseudocode for the Spark-based implementation of PAS3-HSID can be seen in Algorithm 2, and the source code is available on GitHub[2].

[2] `https://github.com/beccatickle/PAS-HSID`

Algorithm 2: Spark Streaming-based implementation of PAS3-HSID

Require: HotSpotsRDD (from previous interval); Incidents; DecayRate; DeleteThreshold; HotSpotThreshold; MileageRange; InitNumHotSpots; NumPartitions

$IncidentsDStream \leftarrow$ **textFile**$(Incidents)$
$IncidentPairs \leftarrow IncidentsDStream$.**map**$(i \Rightarrow <(lat_i, lng_i), infoArr_i>)$
STAGE 1
if $HotSpotsRDD$.isEmpty **then**
  $HotSpotsBC \leftarrow$ **broadcast**$(IncidentPairs$.**takeSample**$(InitNumHotSpots))$
else
  $HotSpotsBC \leftarrow$ **broadcast**$(HotSpotsRDD$.**collect**$())$
end if
$ReducedIncidents \leftarrow IncidentPairs$.**mapPartitions**$(data \Rightarrow$ ReduceWithHotSpots$(data$, $MileageRange$, $HotSpotsBC))$
$HotSpotUpdates \leftarrow ReducedIncidents$.**filter**$(i \Rightarrow isReduced_i)$.**reduceByKey**$((a, b) \Rightarrow n_a + n_b)$
$RemainingIncidents \leftarrow ReducedIncidents$.**filter**$(i \Rightarrow !isReduced_i)$
STAGE 2A
$NewHotSpots \leftarrow RemainingIncidents$.**mapPartitions**$(data \Rightarrow$ RemoveRedundanciesInPartition$(data$, $MileageRange))$
STAGE 2B
for $i = 0$ to $NumPartitions$ **do**
  $partitionBC \leftarrow$ **broadcast**$(partition_i$.**collect**$())$
  $NewHotSpots \leftarrow NewHotSpots$.**mapPartitions**$(data \Rightarrow$ ReduceWithPartitionI$(data - partition_i$, $partitionBC$, $MileageRange))$
end for
STAGE 3
$HotSpotsRDD$.**map**$(h \Rightarrow FV_h \cdot (1 - DecayRate))$
$IntermediateState \leftarrow HotSpotsRDD$.**union**$(HotSpotUpdates$.**union**$(NewHotSpots))$
$AggregatedFitness \leftarrow IntermediateState$.**reduceByKey**$((a, b) \Rightarrow FV_a + FV_b)$
$NewStateRDD \leftarrow AggregatedFitness$.**filter**$(h \Rightarrow FV_h > DeleteThreshold)$.**cache**$()$
return $NewStateRDD$.**filter**$(h \Rightarrow FV_h > HotSpotThreshold)$

Stage 1 of the algorithm identifies those incidents that can be reduced by existing hot spots, and are therefore redundant and can be removed. The current set of hot spots is stored as an RDD and so is distributed across nodes. This is also true of the RDD containing the incidents for the present interval, which is created by reading from a streaming source. Here, we simply load newly arrived incidents from a text file, but any Spark input source could be used.

In order for all hot spots to be available at each node, they must first be collected back to the driver, before being broadcast to all workers. Typically, the set of hot spots should be small enough that it can be held in main memory of all the worker nodes. When the set of hot spots is empty at the start of Stage 1, we precede this with a *takeSample* operation that collects 10% of the incidents back to the driver node to form an initial hot spot set. We then sequentially remove any redundancies within this set before broadcasting it.

The Spark transformation *mapPartitions* is then used to apply a function *ReduceWithHotSpots* to each partition of the incidents RDD. This function iterates through the incidents within a single partition, comparing them to the broadcast hot spots. If an incoming incident is similar enough to any existing hot spot, then that incident's information is effectively replaced by the hot spot's key and value, although with a count $n_h = 1$ instead of the fitness value. This signifies a single incident being added to the hot spot. Details of this function can be found in Algorithm 3. Any incidents that are not successfully reduced by a hot spot maintain their own information.

The resulting RDD is then split using two *filter* operations to separate the hot spot updates and the remaining incidents. The remaining incidents are operated on in

Algorithm 3: The *ReduceWithHotSpots* function for a partition P

Require: Incidents_P; HotSpotsArr; MileageRange

$result \leftarrow []$
for all i in $Incidents_P$ **do**
  if there exists h in $HotSpotsArr$ similar enough to i **then**
    $result \leftarrow result + < key_h, (infoArr_h, n_h = 1, isReduced_h = true) >$
  else
    $result \leftarrow result + < key_i, (infoArr_i, n_i = 1, isReduced_i = false) >$
  end if
end for
return $result$

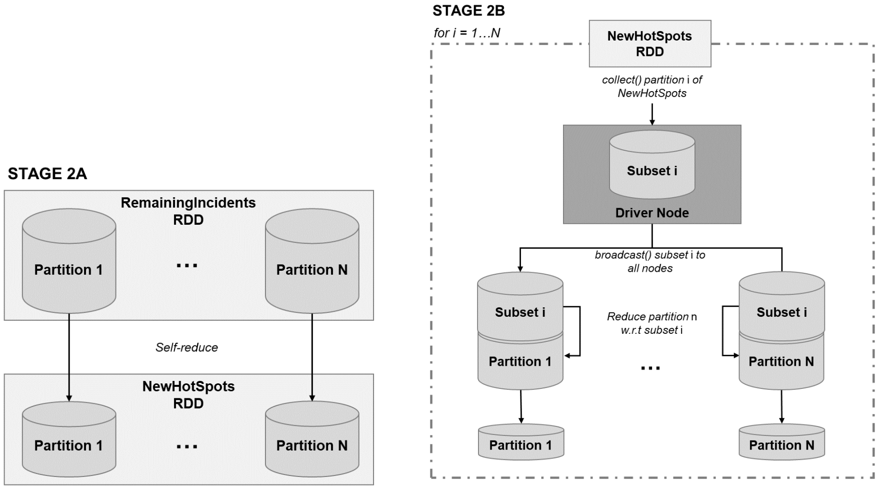
the next stage, whilst the hot spot updates undergo a *reduceByKey* operation. The RDD *reduceByKey* function is similar to the MapReduce *reduce*; however, instead of returning a single value which is the result of combining all items of an RDD in some way, *reduceByKey* returns one value per key that exists in the RDD. Here, the count n_h is accumulated for each key (i.e. each hot spot h), representing the number of incidents reduced by each h. This creates an RDD containing the keys of existing hot spots that have reduced incidents in this interval, alongside the number of such incidents. This information is used in Stage 3 to update the state.

We present two different implementations of Stage 2, a decision also taken in [31]. The first is a sequential version, that makes the assumption that the majority of incidents are reduced in Stage 1. This is tested later in the experimental study to establish if it is a valid assumption to make. Therefore, the set left over to be reduced is sufficiently small to collect back to the driver and operate on sequentially. Each incident i is compared against all other incidents, until one that is close enough is found, at which point i is removed and the fitness value of the corresponding incident (now established as a hot spot centre) is incremented to keep track of the number of incidents it includes. Incidents that are unable to be reduced become hot spot centres in their own right, with an initial fitness value of 1.

The alternative version of Stage 2 parallelises the computation. This version performs more efficiently when the set of incidents left over is very large and would take too long to process in a sequential manner. First, each partition of the RDD is reduced individually in a similar way to the sequential version, identifying hot spot centres within individual partitions (Figure 5a). We then iterate through the partitions one by one (Figure 5b). At each iteration, the current suppressing partition is broadcast to all nodes. All other partitions are then reduced with respect to the hot spots contained within the suppressing partition, removing individual incidents as appropriate, if a hot spot is found close enough.

By the end of Stage 2, two RDDs have been formed which together contain all the information necessary to update the state. One contains keys of already existing hot spots that have had incidents added to them within the current interval (formed in Stage 1), whilst the other contains keys of newly identified hot spot centres. Both RDDs also store the number of incidents n_h reduced by each hot spot this interval.

The third stage involves the combination of these two RDDs with the current state RDD, to generate an RDD containing the new state. The fitness values for each hot spot key are calculated according to the formula given in Subsection 3.1, with the first step being to decay the fitness of the current hot spots by the specified decay rate. *Union* operations are then used to join the current state with the two RDDs produced in Stages 1 and 2, before a *reduceByKey* operation is performed. The



(a) (b)

Fig. 5: Parallelisation of Stage 2 of PAS3-HSID. Partitions are reduced individually (a), before being iteratively reduced with respect to the other partitions (b). Note how the size of the partitions reduces throughout Stage 2 as redundant incidents are removed.

reduce function provided sums the fitness values for identical keys, thus increasing the fitness value of each hot spot by n_h. The initial fitness value for a newly identified hot spot is therefore simply the number of incidents that it covers.

The final step for updating the state is to remove those hot spots with a fitness value less than a given deletion threshold, achieved using a *filter* operation. The resulting RDD represents the new state and is cached in memory so that it can be efficiently accessed at the next time interval. In order to determine the set of hot spots to return as the output for this interval, the state RDD can be further filtered to leave only those hot spots with a fitness value that is greater than the specified hot spot threshold. This set can then be saved to files as required.

4 Experimental Study

In this section, we investigate and assess the behaviour of the proposed PAS3-HSID algorithm. To do this, we first define the following experimental set-up. We employ two different sets of telematics data for HGV incidents within the UK. Initially, we were provided with data from a three-month period covering speeding, harsh cornering, harsh braking and contextual speeding incidents. The speeding and contextual speeding categories differ in how the speed limit is determined. For speeding incidents, vehicles have exceeded the limit specified by the road signs. For contextual speeding, other factors are also taken into account. For example, if it is raining, then HGVs may be required to travel slower due to the road being wet. The location, bearing, address and date/time of occurrence are given for each incident.

We have split these datasets into batches covering various time periods, namely 12 hour-long, day-long and week-long batches. Additionally, they were also each split into ten equally-sized batches. Dividing the data in this way allows us to examine the behaviour of the algorithm with a variety of batch sizes, ranging from very small to much larger. Table 1 shows the average number of incidents per batch for each batch length and dataset.

Table 1: Average number of incidents per batch for the original datasets

Dataset	12 Hours	Day	Week	Equal	Total
Speeding	17	34	230	313	3139
Harsh Cornering	73	146	970	1359	13592
Harsh Braking	1149	2298	16032	21369	213696
Contextual Speeding	3881	7762	53453	72187	721878

A larger dataset collected over nine months is also employed to further assess the robustness of the method. This contained speeding, harsh cornering and harsh braking incidents, with 2,283,305, 2,285,088 and 4,515,990 data points, respectively. Analysis of this dataset shows that the number of incidents each day is no greater than in the original dataset, and so we decided to only split these into ten batches of equal size to enable an evaluation of how the algorithm performs running on a cluster with larger batch sizes.

When characterising the behaviour of the algorithm, we discuss both the runtime and the influence of a variety of parameter settings on the number of hot spots retained. The parameters considered are shown in Table 2. These values have been empirically adjusted over a number of preliminary experiments. Each batch is partitioned into a number of partitions, located on different nodes. From the runtime perspective, we test various numbers of partitions in the experiments carried out on the cluster. In addition to this, we compare the two implementations of Stage 2 of the algorithm (sequential and parallel) in order to establish in which scenarios it is best to pick one implementation over the other.

Table 2: Parameter values investigated in the experiments.

Parameter	Values tested
dr	0.1, 0.3, 0.5
$delTh$	0.9, 1.9
$hsTh$	3, 5, 7
$\#partitions$	4, 8, 12, 24, 48

We also aim to show the advantages of our pheromone-based algorithm in comparison to other HSID approaches. Due to the lack of methods available in the literature for HSID on big data streams, we are limited in the comparisons we can make. We therefore focus on the differences between PAS3-HSID, with its pheromone mechanism for determining hot spots and their relevance, and the original SeleSup HSID algorithm, without such a mechanism. We use two alternative ways of applying SeleSup HSID to the data for the comparison, namely:

- Applying SeleSup HSID to each dataset as a whole, allowing us to compare against a HSID method that does not account for hot spots changing over time. We refer to this approach as SeleSup-HSID-D.
- Applying SeleSup HSID to each streaming interval individually. This enables comparison with a method that should identify changes over time, but without a way of considering previous hot spots when establishing the current hot spot. We refer to this approach as SeleSup-HSID-I.

The parameters chosen for these experiments are displayed in Table 3. Mileage ranges of interest were given and their effect discussed in [31] as {0.5, 2, 5} miles for contextual speeding and speeding data, and {0.1, 0.2, 0.5} for harsh cornering and harsh braking. We run our experiments with the greatest mileage from each of these sets: 5 miles for the speeding datasets, and 0.5 miles for harsh braking and cornering.

Table 3: Parameters used for all the algorithms involved in the comparison experiments.

Algorithm	Parameters
PAS3-HSID	$dr = 0.3$, $delTh = 1.9$, $hsTh = 5$
SeleSup-HSID-I	Percentage of Initial Points = 10, Hot Spot Threshold = 5
SeleSup-HSID-D	Percentage of Initial Points = 10, Hot Spot Threshold = 5

Due to the random component of PAS3-HSID and SeleSup HSID, where an initial set of hot spots is established by randomly selecting a given percentage of the newly arrived incidents, the behaviour of the algorithm can differ when presented with the same data. Therefore, we run all experiments twenty times, and average the results of all executions.

The experiments with the original datasets (3 months data from [15,31]) have been carried out in a single node with an Intel(R) Xeon(R) CPU E5-1650 v4 processor (12 cores) at 3.60GHz, and 64 GB of RAM. In terms of software, we have used the Cloudera's open-source Apache Hadoop distribution (Hadoop 2.6.0-cdh5.14.2) and Spark 2.0.0. In our experiments, we have set a total number of 8 concurrent tasks. The experiments on the larger datasets have been carried out in a cluster composed of 14 computing nodes managed by the master node. Each one of these nodes has 2 Intel Xeon CPU E5-2620 processor, 6 cores (12 threads) per processor, 2 GHz and 64 GB of RAM. The network is Infiniband 40Gb/s. This hardware was configured to provide a maximum number of current tasks to 256. In terms of software, every node runs on Cent OS 6.5, and uses Cloudera's open-source Apache Hadoop distribution (Hadoop 2.6.0-cdh5.8.0) and Spark 2.2.1.

The following subsections present the results of these experiments. Subsection 4.1 discusses the impact of different parameter choices on the behaviour of the algorithm, whilst in Subsection 4.2 we perform a comparison with alternative HSID approaches. Finally, Subsection 4.3 covers the experiments executed on the cluster.

4.1 Analysis of algorithm behaviour

PAS3-HSID has multiple parameters that can be altered in order to influence the hot spots being identified. There is no exact measurement of what amounts to a satisfactory number of obtained hot spots. Instead, our aim here is to provide a detailed analysis of how parameter choices can impact upon the behaviour of the algorithm. We discuss the effects of the decay rate, delete threshold and hot spot threshold parameters, which are further introduced and discussed below. These experiments have been run using eight partitions, although this should have no impact on the behaviour of the algorithm in terms of hot spots identified.

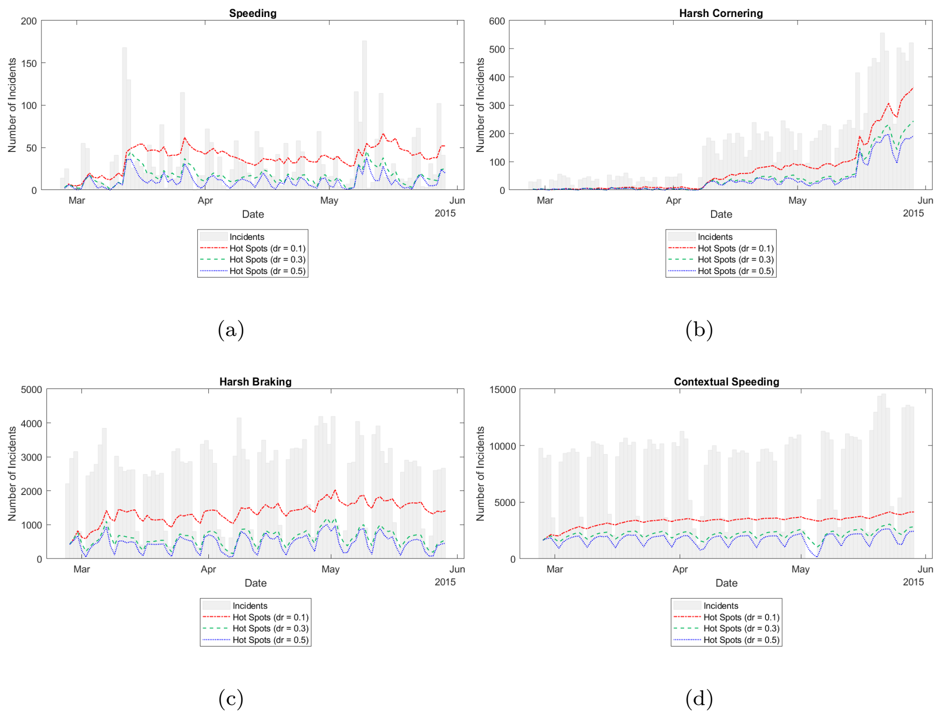


Fig. 6: Comparison between different decay rates for PAS3-HSID. The datasets used were split into daily batches of incidents, and the experiments were run with 8 partitions.

The effect of setting different decay rates (0.1, 0.3 and 0.5) when the algorithm is applied to daily batches is shown in Figure 6, alongside the distributions of the original incidents. We can observe that for datasets with a more regular pattern of incidents, the number of hot spots that are identified increases throughout each week before decreasing over the weekends when there are naturally fewer incidents. The method is also able to adapt quickly to the sudden changes in the irregular distribution of the speeding dataset. A decay rate of 0.1 seems to be suitable for the smaller datasets; however, for the contextual speeding data it results in a general increase in hot spots over time, suggesting that old hot spots are not forgotten quickly enough. A rate of 0.3 is able to handle a short period of time with very few incidents, such as the few days in early May (Figure 6d) where there was a sudden decrease in batch size for contextual speed; 0.3 resulted in a larger proportion of previous hot spots being retained over these days than 0.5, which lost the majority of all hot spots that were stored.

The algorithm relies on two thresholds relating to the fitness of hot spots. The first, the delete threshold $delTh$, establishes at what point it is no longer worth storing a hot spot in the state, resulting in its deletion. The hot spot threshold $hsTh$, determines when we would consider a hot spot to be significant enough for us to know about. Such hot spots are returned at the respective streaming interval, and can subsequently be visualised or further processed if required. Figure 7 shows one such visualisation of hot spots identified over five days in a small region for the harsh braking dataset. We can observe how some hot spots with a low fitness value are only present for a short time, whilst those with a large fitness value, representing a consistently high number of incidents, are more long-term.

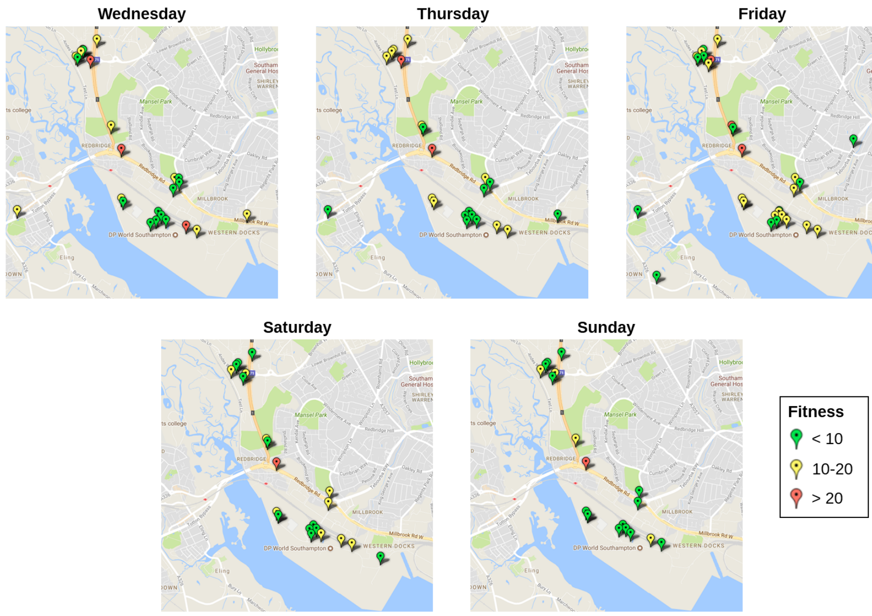


Fig. 7: Hot spots obtained by PAS3-HSID in Southampton, UK over a five-day period in April 2015, showing fitness values changing and the addition and deletion of hot spots over time. (Harsh braking data split into daily batches, $hsTh$=3)

For these experiments, we have tested various values for both thresholds. For the hot spot threshold, it is clear that increasing the threshold results in fewer hot spots being returned at a given time interval whilst the number of hot spots in the state is unaffected (Table 4). Values chosen for $hsTh$ may vary depending on the approximate expected size of incident batches. For example, for very small batches (speeding and harsh cornering datasets), a lower threshold is likely to be preferable due to fewer hot spots, generally with lower fitness values. Conversely, a higher value for $hsTh$ is more suitable for the harsh braking and contextual speeding data, as with a high number of hot spots stored in the state, those with lower fitness values are less significant.

Table 4: Number of hot spots found for various hot spot thresholds, averaged per streaming interval. The datasets are split into week-long batches.

Dataset	$hsTh$	$FV > delTh$	$FV > hsTh$
Speeding	3	82	54
	5	83	33
	7	82	23
Harsh Cornering	3	246	133
	5	246	65
	7	246	40
Harsh Braking	3	5361	3245
	5	5360	1596
	7	5356	931
Contextual Speeding	3	6275	5106
	5	6273	3902
	7	6276	3222

The delete threshold directly impacts the hot spots that are maintained in the state between streaming batches. Figure 8 shows the effect of two delete thresholds (0.9 and 1.9) on both the number of hot spots in the state, and the number with a fitness greater than a hot spot threshold of 5. It can be seen that increasing the value of $delTh$ to 1.9 considerably reduces those in the state, while having a relatively small impact on the number with $FV > hsTh$; this behaviour is consistent across all datasets. The main difference between these threshold values is that 0.9 will keep isolated incidents that could not be allocated to a hot spot within the interval in which they arrive, thus giving them a chance to become a hot spot later. Alternatively, using 1.9 ensures that any isolated incidents are removed within the same interval that they arrive. From this, we can conclude that the majority of these isolated incidents do not subsequently become hot spots. We suggest a delete threshold of 1.9 so that such incidents are removed immediately, resulting in a smaller state being maintained between batches.

Table 5: Average runtime per streaming interval, in seconds, for each dataset and split. $\#partitions$=8, dr=0.3, $delTh$=1.9, $hsTh$=5.

Dataset	12 Hours	Day	Week	Equal
Speeding	0.449	0.412	0.509	0.561
Harsh Cornering	0.430	0.420	0.656	0.800
Harsh Braking	0.703	0.952	4.498	6.161
Contextual Speeding	0.975	1.279	4.164	5.582

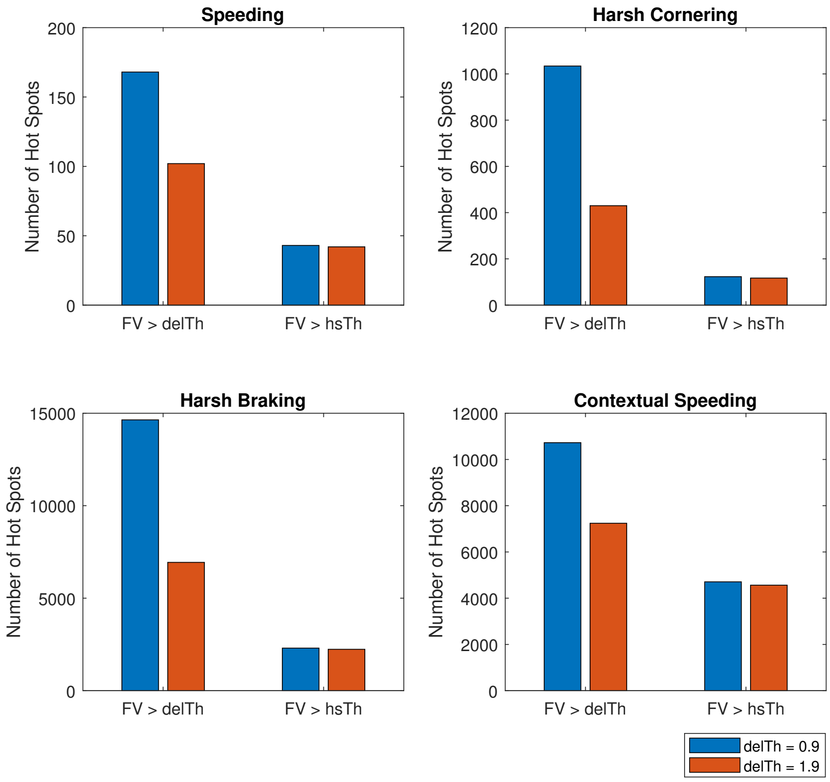


Fig. 8: Comparison of hot spots detected by PAS3-HSID with different $delTh$ values. Note that Number of Hot Spots refers to the average number per streaming interval. Datasets are split into ten equal size batches.

Table 5 shows the average runtime per streaming interval for all datasets and splits used in the experiments. We can observe that all batch sizes included here are processed in a short time. In some cases, contextual speeding batches are processed quicker than harsh braking, despite having more than three times the number of incidents per batch. This is due to different mileages used to define hot spots for these incident types, a behaviour also observed in [31].

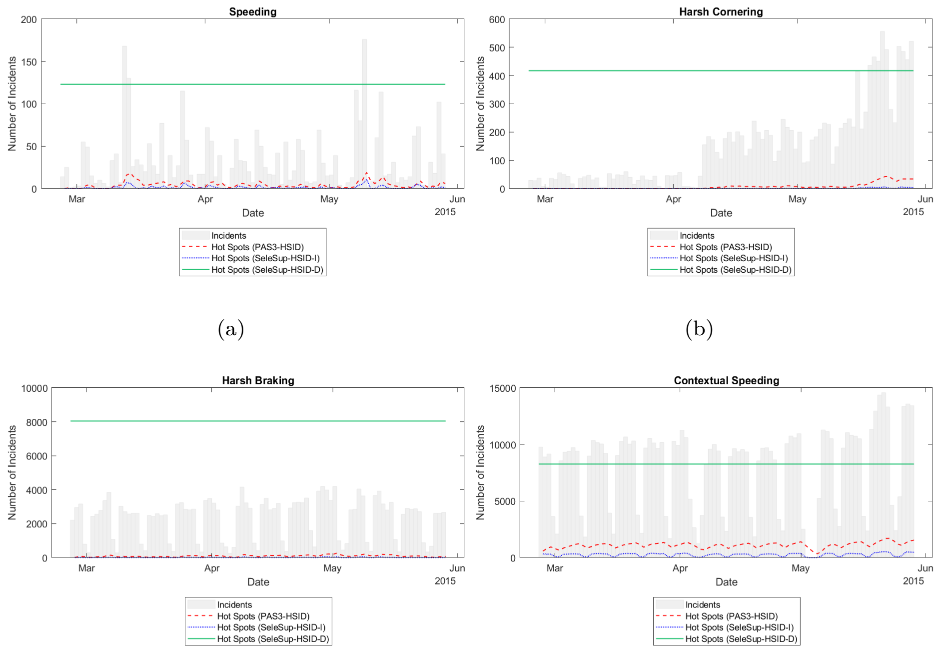
From the results presented here, we can conclude:

- When run on a single node, our Spark-based implementation can efficiently process batches containing tens of thousands of incidents.
- The algorithm can quickly adapt over time, detecting a number of hot spots that is representative of the current incidents.
- The choice of parameters should depend on the data at hand, including rough estimates of the batch size expected in general. For example, smaller decay rates and threshold values are more likely to be suitable for batches containing fewer incidents, and vice versa.

– Future work could perhaps look at incorporating some automatic adaptation of these parameters as the algorithm runs, in response to changes in the nature of the data arriving. This would avoid them having to be fixed at the start of execution.

4.2 Comparison with other methods

We now compare our algorithm to the two alternative approaches defined above (SeleSup-HSID-D and SeleSup-HSID-I), to show the differences achieved by incorporating some mechanism to maintain hot spot information across streaming intervals into the HSID process for data streams.



(a) (b) (c) (d)

Fig. 9: Incident distributions and hot spots found by PAS3-HSID and the two comparison approaches, SeleSup-HSID-D and SeleSup-HSID-I.

The results in terms of hot spots obtained are shown in Figure 9. From these plots, we can conclude:

– SeleSup-HSID-D clearly provides no information regarding how hot spots change over time. In contrast, PAS3-HSID adapts and the number of hot spots obtained reflects changes in the incident distribution.

- Whilst SeleSup-HSID-I does provide some indication of the dynamic nature of hot spots, it uses no input from previous intervals when establishing the current set of hot spots, therefore finding fewer than our algorithm. On days when there are very few incidents, it is unable to identify any hot spots at all; this is frequently seen on weekends.
- There is a significant reduction in the amount of data kept by our algorithm in comparison to SeleSup-HSID-D, visible on the plots by how much higher the green lines are.

4.3 Performance on larger datasets

Here, we present the results of the experiments performed on the larger datasets and executed on a cluster, as detailed previously. We vary the number of partitions used in order to understand the influence they have on the runtime, as well as comparing the two alternative implementations of Stage 2.

Table 6: Results in terms of number of hot spots obtained (averaged per interval) for the fully parallel version of PAS3-HSID. The datasets used are the larger datasets, each split into ten batches of equal size.

Dataset	$FV > delTh$	$FV > hsTh$
Speeding	6374	3460
Harsh Cornering	36343	18624
Harsh Braking	64493	31514

The average number of hot spots found per interval is shown in Table 6. Despite containing a relatively similar number of incidents per batch, speeding and harsh cornering give significantly different numbers of hot spots; this is due to these datasets each having a different specified mileage range. As shown in [31], a larger mileage reduces the hot spots identified, due to each hot spot covering a larger section of the road.

Figure 10 displays the average runtime per interval of the two different implementations of the algorithm: one where Stage 2 is done sequentially, and one where it is parallelised (fully parallel). We can observe that there is a significant reduction in the runtime when the fully parallel version is used for very large batch sizes, such as in the harsh braking data. In Figure 11 we provide further details of the runtimes of the various algorithm stages, focusing on harsh braking as the largest dataset. It can be seen that when Stage 2 is implemented in a sequential manner, it dominates the overall runtime. Parallelising Stage 2 speeds it up to the extent that it takes less time than Stage 1. Increasing the number of partitions reduces the runtime, although we do observe the beginning of a plateau, suggesting that using a greater number of partitions would not give much performance gain, at least for a dataset of this size. We can conclude that the Spark-based implementation of our proposed algorithm is capable of efficiently handling batches containing hundreds of thousands of incidents, and we advise employing the fully parallel implementation in such scenarios.

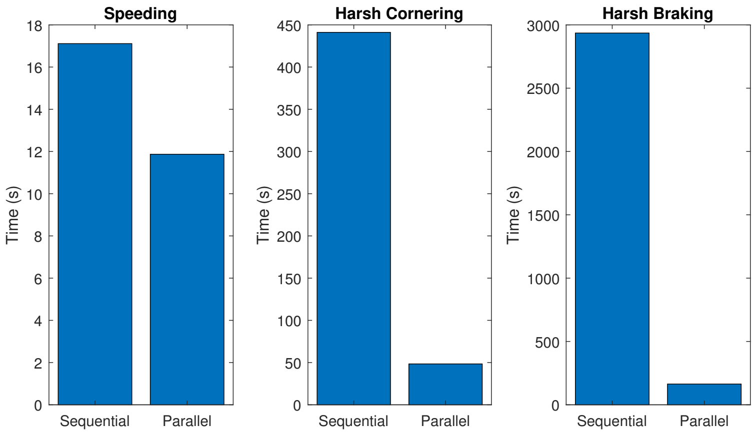
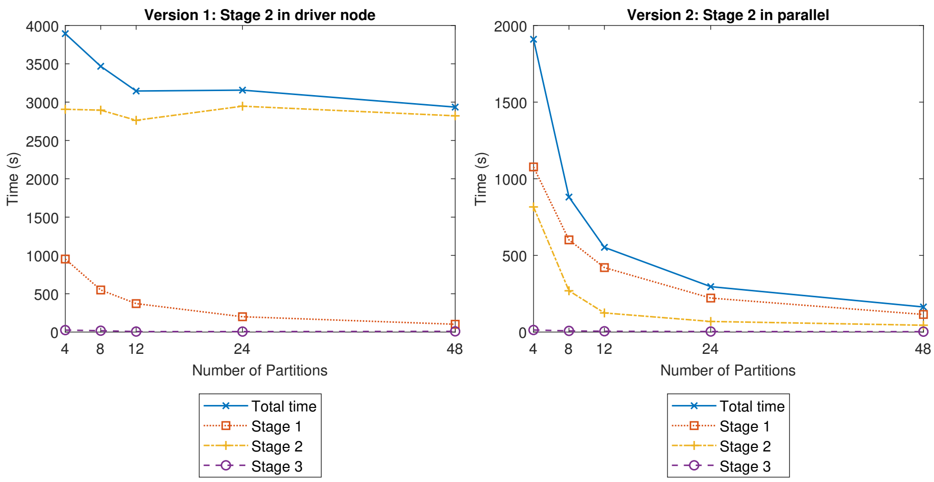


Fig. 10: Comparing the average runtime per interval for the two alternative versions of PAS3-HSID: one with sequential Stage 2 and one with parallel Stage 2. The parallel version shows a significantly better runtime, particularly on larger batch sizes. These experiments were executed with 48 partitions.



(a) (b)

Fig. 11: Comparing average runtimes per interval for each stage of PAS3-HSID, for the two different implementations of Stage 2. Results are obtained for the large harsh braking dataset.

5 Conclusions

In this work we have presented an approach for vehicle hot spot identification in data streams, adapting an existing instance selection method, SeleSup, with a pheromone-based mechanism that ensures the hot spots found are reflective of the recent incident distribution. Our experiments have shown that our bio-inspired approach successfully determines hot spots within a dynamic stream, and when implemented in Apache Spark Streaming is capable of processing large batch sizes of hundreds of thousands of incidents in a timely manner. Furthermore, the algorithm successfully reduces the volume of data retained at each interval of the stream, decreasing storage requirements.

Hot spot identification can be of use in several areas, such as those mentioned at the start of this paper. Further analysis of the applicability of the PAS3-HSID algorithm to domains other than transportation is of interest for future work. Additionally, there are possibilities for improvements to be made to the method itself. For example, we have already mentioned in the previous section the inclusion of automatic parameter adaptation. In terms of the HGV incident scenario specifically, further investigation into supplementary conditions for defining hot spots could be done. For instance, taking into account that number of lorries in geographical regions in order to determine suitable localised thresholds.

6 Compliance with Ethical Standards

Conflict of Interest: The authors declare that they have no conflict of interest.

Ethical Approval: This article does not contain any studies with human participants or animals performed by any of the authors.

7 Acknowledgements

The authors would like to thank the Soft Computing and Intelligent Information Systems research group from the University of Granada, for allowing us to use their big data infrastructure to carry out the experiments.

References

1. Alpaydin, E.: Introduction to Machine Learning. The MIT Press (2014)
2. Anderson, T.K.: Kernel density estimation and k-means clustering to profile road accident hotspots. Accident Analysis & Prevention **41**(3), 359 – 364 (2009)
3. Beringer, J., Hüllermeier, E.: Efficient instance-based learning on data streams. Intelligent Data Analysis **11**(6), 627–650 (2007)
4. Braithwaite, A., Li, Q.: Transnational terrorism hot spots: Identification and impact evaluation. Conflict Management and Peace Science **24**(4), 281–296 (2007)
5. Cambria, E., Chattopadhyay, A., Linn, E., Mandal, B., White, B.: Storages are not forever. Cognitive Computation **9**(5), 646–658 (2017)
6. Cheng, W., Washington, S.P.: Experimental evaluation of hotspot identification methods. Accident Analysis & Prevention **37**(5), 870–881 (2005)
7. Dean, J., Ghemawat, S.: Mapreduce: A flexible data processing tool. Communications of the ACM **53**(1), 72–77 (2010)
8. Ding, S., Zhang, J., Jia, H., Qian, J.: An adaptive density data stream clustering algorithm. Cognitive Computation **8**(1), 30–38 (2016)
9. Dorigo, M., Di Caro, G.: Ant colony optimization: a new meta-heuristic. In: Evolutionary Computation, 1999. Proceedings of the 1999 Congress on, vol. 2, pp. 1470–1477. IEEE (1999)

10. Dorigo, M., Maniezzo, V., Colorni, A.: Ant system: optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) **26**(1), 29–41 (1996)
11. Elen, B., Peters, J., van Poppel, M., Bleux, N., Theunis, J., Reggente, M., Standaert, A.: The aeroflex: A bicycle for mobile air quality measurements. Sensors (Switzerland) **13**(1), 221–240 (2013)
12. Ester, M., Kriegel, H.P., Sander, J., Xu, X., et al.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Kdd, vol. 96, pp. 226–231 (1996)
13. Figueredo, G.P., Ebecken, N.F.F., Augusto, D.A., Barbosa, H.J.C.: An immune-inspired instance selection mechanism for supervised classification. Memetic Computing **4**, 135–147 (2012)
14. Figueredo, G.P., Ebecken, N.F.F., Barbosa, H.J.C.: The SUPRAIC algorithm: A suppression immune based mechanism to find a representative training set in data classification tasks. In: ICARIS, *Lecture Notes in Computer Science*, vol. 4628, pp. 59–70. Springer (2007)
15. Figueredo, G.P., Triguero, I., Mesgarpour, M., Guerra, A.M., Garibaldi, J.M., John, R.I.: An immune-inspired technique to identify heavy goods vehicles incident hot spots. IEEE Transactions on Emerging Topics in Computational Intelligence **1**(4), 248–258 (2017)
16. Gama, J.: Knowledge Discovery from Data Streams, 1st edn. Chapman & Hall/CRC (2010)
17. García, S., Derrac, J., Cano, J., Herrera, F.: Prototype selection for nearest neighbor classification: Taxonomy and empirical study. IEEE Transactions on Pattern Analysis and Machine Intelligence **34**(3), 417–435 (2012)
18. García, S., Luengo, J., Herrera, F.: Data Preprocessing in Data Mining. Springer Publishing Company, Incorporated (2014)
19. Han, J., Kamber, M., Tung, A.K.H.: Spatial clustering methods in data mining: A survey. In: H.J. Miller, J. Han (eds.) Geographic Data Mining and Knowledge Discovery, Research Monographs in GIS. Taylor and Francis (2001)
20. Han, J., Pei, J., Kamber, M.: Data mining: concepts and techniques. Elsevier (2011)
21. Hulten, G., Spencer, L., Domingos, P.: Mining time-changing data streams. In: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '01, pp. 97–106. ACM, New York, NY, USA (2001)
22. Klinkenberg, R.: Learning drifting concepts: Example selection vs. example weighting. Intelligent Data Analysis **8**(3), 281–300 (2004)
23. Krempl, G., Žliobaite, I., Brzeziński, D., Hüllermeier, E., Last, M., Lemaire, V., Noack, T., Shaker, A., Sievi, S., Spiliopoulou, M., Stefanowski, J.: Open challenges for data stream mining research. SIGKDD Explor. Newsl. **16**(1), 1–10 (2014)
24. Mesgarpour, M., Landa-Silva, D., Dickinson, I.: Overview of telematics-based prognostics and health management systems for commercial vehicles. Activities of Transport Telematics **395**, 123–130 (2013)
25. Montella, A.: A comparative analysis of hotspot identification methods. Accident Analysis & Prevention **42**(2), 571 – 581 (2010)
26. Passini, M.L.C., Estébanez, K.B., Figueredo, G.P., Ebecken, N.F.F.: A strategy for training set selection in text classification problems. International Journal of Advanced Computer Science and Applications **4**(6), 54–60 (2013)
27. Perallos, A., Hernandez-Jayo, U., Onieva, E., García-Zuazola, I.J.: Intelligent Transport Systems: Technologies and Applications, 1st edn. Wiley Publishing (2015)
28. Ramírez-Gallego, S., Krawczyk, B., García, S., Woźniak, M., Herrera, F.: A survey on data preprocessing for data stream mining: Current status and future directions. Neurocomputing **239**, 39 – 57 (2017)
29. Shirkhorshidi, A.S., Aghabozorgi, S., Wah, T.Y., Herawan, T.: Big data clustering: a review. In: International Conference on Computational Science and Its Applications, pp. 707–720. Springer (2014)
30. Siddique, N., Adeli, H.: Nature inspired computing: An overview and some future directions. Cognitive Computation **7**(6), 706–714 (2015)
31. Triguero, I., Figueredo, G.P., Mesgarpour, M., Garibaldi, J.M., John, R.I.: Vehicle incident hot spots identification: An approach for big data. In: 2017 IEEE Trustcom/BigDataSE/ICESS, pp. 901–908 (2017)
32. Van Brummelen, G.: Heavenly mathematics: The forgotten art of spherical trigonometry. Princeton University Press (2012)

33. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, NSDI'12, pp. 15–28 (2012)
34. Zaharia, M., Das, T., Li, H., Shenker, S., Stoica, I.: Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters. In: Proceedings of the 4th USENIX conference on Hot Topics in Cloud Computing, pp. 10–10 (2012)
35. Zhao, L., Wang, L., Xu, Q.: Data stream classification with artificial endocrine system. Applied Intelligence **37**(3), 390–404 (2012)