LSDE13: Shipping Safety

Radu Jica and Tim Visser

Abstract— **Find almost-collisions in the English channel and visualize on an interactive map. This is done by reconstructing ship paths and distance computation in latitude, longitude, and time dimensions.**

I. Introduction

Over 70 percent[1] of global trading is carried out via international shipping transports, therefore shipping safety is an important topic to study and to maintain. In this project, we analyze 2 data-sets of AIS [1] messages with the purpose of detecting near-collisions between ships. With the first data-set comprising of 14 days-worth of AIS messages and the second with over 2 years of data, one is able to apply a variety of algorithms and different filtering methods to detect near-collisions.

In order to display these results, the second part of this project involves creating a static interactive visualization of near-collisions. This visualization will allow for the intuitive interpretation of relevant results in time ranges of particular interest to the user.

II. Related Work

In their research, Zhang et al. [3] describe a multi-factor approach of detecting near miss ship collisions from AIS data. Their method takes several factors into account including the distance between two ships, their respective headings and relative speed. Teixeira et al. [4] reviewed several different methods of collision detection. They found that representing involved ships as projected rectangles based on their width and length (as specified in AIS messages) yields better results than when detecting potential near-collisions using a radius imposed on a ship's center. Others like Wang et al. [5] and Silveira et al. [6] have looked at the same problem in ways that are less relevant to our scope.

[1] Automatic Identification System

III. Research Questions

Within our project, we set out to investigate and answer several research questions pertaining to our topic. These questions are as follows:

- How to process data over a timeline efficiently?
- How to detect collisions effectively (is checking every 1 min enough?)?
- How reliable is AIS data for our purposes?
- How can results from our experiments be visualized intuitively?

By providing answers to these questions we will produce the end result of an interactive visualization of near-collisions between ships within the English channel.

IV. Project Setup

The way we approached the project can be split into 3 main steps, further explained below.

A. Data Comprehension

First and foremost, we needed to understand the data. We have 2 data-sets, first in text format and gzipped, second in just text format. They are grouped as seen in Figure 1 on SurfSARA's HDFS. The data itself comprises of AIVDM sentences[2] grouped by unix timestamps in millisecond precision, as seen in Figure 2. Each file contains roughly 60.000 lines, meaning there are over 3 million entries to process per hour. Putting aside the AIVDM protocol itself, of importance to us are the AIS messages encoded within it. As seen in Figure 2, the AIS messages are encoded in the long strings of characters in the 5th position after '!AIVDM'.

The AIS messages can be of 27 types [2] but after looking through what they contain, we discover that the most relevant types are 1, 2, 3 and type 5. Types 1, 2, and 3 are Position Reports

[2] The messages are emitted by receivers for AIS.

```
/lsde/ais2/2014/10/06/12-18.txt
/lsde/ais2/2014/10/06/12-19.txt
/lsde/ais2/2014/10/06/12-20.txt
/lsde/ais2/2014/10/06/12-21.txt
/lsde/ais2/2014/10/06/12-22.txt
/lsde/ais2/2014/10/06/12-23.txt
```

Fig. 1. Dataset File Structure

```
1443650400.010568 !AIVDM,1,1,,B,15NOHL0P00J@uq6>h8Jr6?vN2>`<,0*4B
!AIVDM,1,1,,A,4022051uvOFD>RG7kDCmliW0088i,0*23
...
1443650400.022010 !AIVDM,1,1,,B,13@uip0v1NOvwBVFS=g:u`rL0>`<,0*64
!AIVDM,1,1,,B,13aDqi0P00PF7kIMw8L@0?vN2>`<,0*0D
...
```

Fig. 2. Database File Content

of Class A containing information such as location, speed over ground, and heading. These messages are broadcasted every 2 to 10 seconds, depending on the ship speed, or every 3 minutes, when the ship is anchored or stationary. Type 5 messages contain ship-specific data, of relevance being ship dimensions in the 4 directions, measured from the antenna to the sides of the ship. Two last types are relevant, namely 18 and 19, containing Position Reports of Class B transmitters. However, these were not used due to constraints from our decoder, detailed in further sections. All of the above ships are identified through their MMSI opening up the path to interesting data analysis because of the MMSI[3] containing information about the country of origin.

Furthermore, it was important to understand the distribution of the data and perhaps to observe some patterns. For this purpose, we have plotted 1 hour of data on the map, as seen in Figure 3. Notice that ships are mostly detected close to the coast and not far at sea. We have also decoded the first digits of the MMSI's for this hour of data to determine from which countries the ships originate. As seen in Table I, most of the ships are from the Netherlands, followed by the United States of America, Canada and several primarily European companies. This might indicate that the data was gathered in a way that presents a bias towards these listed origins.

[3]Maritime Mobile Service Identity. However, the MMSI is not unique for a ship as a ship may have multiple MMSI in its lifetime, in cases such as changing countries. The only unique number, given at build-time, is the IMO identification.

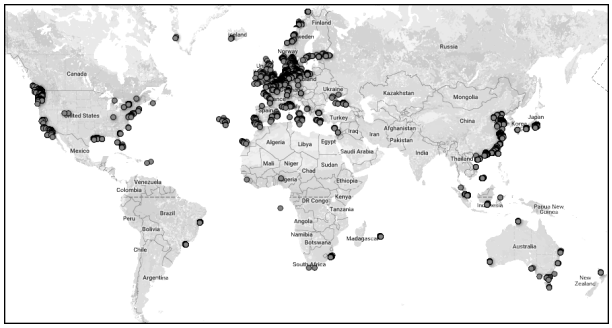

Fig. 3. AIS message location distribution

TABLE I
SHOWING THE RELATIVE OCCURRENCE OF SHIP ORIGINS SAMPLED FROM 1 MINUTE OF DATA

Origin	Count
The Netherlands	13865
United States of America	6618
Canada	5116
Germany	2377
Belgium	2313
Sweden	2102
United Kingdom	2072
Japan	2055
China	2029
Denmark	1454

B. Algorithm selection

The second step involved deciding on which near collision algorithm(s) we want to use and therefore, which data we need to filter for. This has been done by looking into related work, as previously described; however, our decision was to first implement a simple algorithm for near-collisions before attempting something more intricate. One of or the simplest algorithm to detect near-collisions is to have radii around each ship; first through a magic number, namely 5km[4][7], and second through a parameter based on the ship's size. Both these algorithms have been attempted, in this order. The next step is to implement an algorithm presented in related work, namely by Silveira et al. [6], involving vector projections. However, due to time constraints, this has not been fulfilled.

[4]This value was chosen based on the International Regulations for Preventing Collisions at Sea

C. Analysis

The last step was to choose the tools for out data analysis. We chose SPARK[5], due to the efficient in-memory computations, and Python[6], due to its ease-of-use and previous experience in. Using Spark, we could effectively perform map and filter operations through Spark's RDD's. Furthermore, we had to find an AIS-message decoder, preferably in Python; two were found, however one could not be used on SurfSARA due to its non-available dependencies, therefore we were left with one that is based on research code. The issues with this decoder are further explored in the next section. Lastly, for the interactive visualization, we chose MapBox[7] and leaflet.js[8].

With the tools chosen, one had to design the overall pipeline for the above steps. The way the data is grouped provided a hint on how to read and process it, namely using Spark's wholeTextFiles which keeps track of the file's name and data. This allowed us to do computations on a per-minute basis which turned out to be a good method, further supported by observations in [4]. Then, there were several filtering steps that could reduce our data, perhaps significantly. First, only some AIS message types may be relevant for near-collision detection; second, we are interested only in ships within the English Channel; third, only valid data can be used, i.e. no missing required values such as location; last, stationary ships in ports are not particularly interesting, unless other ships collide into them. Arguably the most important step remaining was the actual near-collision detection, involving distance computations or further more sophisticated algorithms. With these effectuated, the results could then be saved in a file for the visualization. The full pipeline is presented at the end of the next section.

V. Experiments

During our experiments we ran into several problems which we will briefly describe. As mentioned in the previous section, most of these problems arose over time from our selected decoder[9] as provided by the GPSD project. The decoder as-is, classified as being research code, does not always do a good job of handling errors properly, nor parsing multiple messages. On many occasions, an unexpected input would lead to a never-ending loop being triggered. This happened in several different cases, some of these occurring very rarely. We worked around these problems by avoiding some of the error handling provided by the decoder and signaling an error to the calling function directly.

Given that most near-collision algorithms require ship size information, we thought the best approach is to use the Type 5 messages from our data-set. However, we have encountered a major issue: according to [2], most Type 5 messages are split into 2 sentences and our chosen decoder could not process these sentences properly. We were required to create custom functions to handle the data-set line-by-line, hence rendering us unable to decode multiple-line messages. This has presumably removed most of the type 5 messages in our data-set because we were only able to detect around 80 distinct ships in an entire day of data, while there were over 3.000 ships sending Type1-3 messages. Therefore, we chose to use a separate ship-size data-set obtained from Marine Vessel Traffic[10].

We attempted to filter our raw data using the constrains presented in the previous section, namely location, message type, non-missing valid data, and duplicates. The naive assumption was that the data-set contained mostly good information which however proved to not be the case, as seen in Figure 4. Note that 'not-missing keys/values' refers to the following keys, apart from mmsi: longitude, latitude, heading, speed over ground, and navigation status, for type 1-3, and the 4 size-measurements, for type 5; and relevant status as anything but stationary and not-applicable (511). As one can see in the figure, filtering out all the above missing key-values reduced our data-set considerably and using the navigation status was not feasible due to ships not respecting

[5]Particularly, version 1.6.1 available on SurfSARA

[6]Version 2.7.5, as available on SurfSARA

[7]https://www.mapbox.com

[8]http://leafletjs.com

[9]https://fossies.org/linux/gpsd/devtools/ais.py

[10]http://www.marinevesseltraffic.com/2013/12/ships-database.html

original file:	75785
only type1,2,3:	64439
not missing keys/values:	18226
only relevant nav status:	4027
within english channel:	80
remove duplicates:	9

Fig. 4. Number of lines remaining after filtering steps in a single file.

the protocol by either not adding any status, or not changing it accordingly. However, the relevant remaining filters were: type1-3, within English channel, and duplicate removal.

With the above in mind, we designed the following data engineering pipeline which was used on increasingly larger subsets of the data-set, followed by a few design-choice explanations, and some results. Particular details of the pipeline and the inner data formats can be seen in the well-commented code. Conclusions from this pipeline are presented in the last section along with insights from the visualization.

Final pipeline

1) Read data and partition accordingly: sc.wholeTextfiles, split into lines, discard unix timestamps; change filename to yymmddhhmm format[11]
2) Decode: Type 1-3 message data, None otherwise. Initially, type 5 messages were also kept but removed as previously discussed.
3) Filter: missing relevant key-values, location=English channel, invalid locations (511 error code), remove duplicates, cache. Only 1 message per mmsi was kept per minute (file). The remaining data-set was cached for later use.
4) Load ship-size data-set: the data-set contained mmsi/length/beam data which was converted into mmsi/radius data in Python dictionary format for quick access. Lastly, it was broadcast-ed to workers for later use.
5) Ship Pairs and near-collisions: combinations of every 2 ships (mmsi) were created per file (minute) of data. For each pair, near-collisions were tested according to the chosen algorithm.
6) Reduce step: all the pairs of ships involved in near-collisions from the previous step were reduced into list of near-collision involved ships and broadcast-ed.
7) Remove stationary ships: get all entries for the ships above and group by mmsi; remove ships which are found stationary.
8) Save results: the data-set, updated according to the previous step, is saved in csv format for visualization.

[11]year, month, day, hour, minute.

We processed the data on a per-minute basis under the assumption that a near-collision cannot be detected in a smaller time frame. This means that a single entry per mmsi per minute was enough to keep and hence the other entries could be discarded. This approach was supported through the use of Spark's wholeTextFiles keeping both the file name (containing the time stamp) and the data.

Since the mmsi ship data was necessary to be used against the entire data-set and our first near-collision algorithms were based solely on radius, it made sense to preprocess the radius so that it can then be broadcasted. The radius was computed by summing up the length and beam measurements, dividing by 2, and multiplying by a 'magic' parameter. Multiple values have been tested, though one needs some classified data to test which one was better. This step can be skipped depending on the chosen near-collision algorithm.

We have tried 2 different algorithms where pairs of ships were classified as near-collisions: first, if the distance between them was lower than the arbitrary value of 5km, and second, when the sum of their computed radii was larger than the distance between them. We believed the radii approach was better because a smaller ship usually entails more mobility and hence 5km would be too much. In Table II, one may observe the average computed radius size of the ships and how they correlate to the number of detected near-collision. Notice there is a 'sweet spot' between parameter 30 and 40 where the number of near-collisions increases drastically up to a plateau; this can be better visualized in Figure 5.

Lastly, stationary ships were removed based on the assumption that if in a pair of ships, both did

not move more than the arbitrary distance of 50 meters, they are stationary. The reasoning behind it was to remove ships that are close to each-other and hence detected as near-collisions, when in fact are either anchored in ports or fishing close-by. This was implemented by sorting the grouped-by-mmsi entries on longitude and computing the distance between the first and last entry; if the distance was less than 50 meters, the ship has not moved in its lifetime in our data-set. This approach clearly works in a smaller time frame, however in practice on our data, it seemed that there were no stationary ships overall, i.e., even if stationary and transmitting for a couple of hours, the ships moved in the end. This approach can be perfected by performing the check repeatedly on a smaller data-frame, such as every hour.

The largest dataset on which the Python script was run was 3 months of data from 2014. We have used 480 executors and a few other tweaks which can be seen in the code. We argue, based on the duration of 1 hour and 20 minutes for this dataset, that the RDD use and overall design is efficient.

There are 3 Python scripts attached to this submission: *crunch*, processing data without using ship size; *crunchWith5*, using the size information from type 5 messages, and *crunchWithDB*, using the mmsi-size database. The last script is the most in-depth one and used for the results in this paper.

TABLE II

SHOWING THE INCREASING NUMBER OF NEAR-COLLISIONS DETECTED USING THE RADII METHOD IN 1 DAY OF DATA.

Radius Param.	Average radius(m)	Number of near-collisions
10	962	50
20	1924	57
30	2887	64
40	3849	79
50	4812	80

With regard to visualization, we take the following approach:

1) Import and parse CSV of all detected near-collisions
2) Iterate over CSV file, selecting near-collisions in correct date range (jump table used in order to reduce number of iterations)

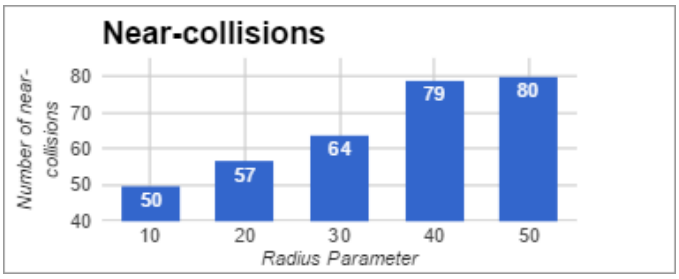

Fig. 5. Plot of Table II to better notice the 'sweet point' for the radius parameter.

3) Generate a random colour which is saved to a lookup table or if already present, look it up
4) Plot the data point as a circle with the colour found in step 3

The user can provide several inputs to this interactive visualization in order to express a degree of control upon it. Primarily, an available slider allows the user to set an interval in which near-collisions are drawn onto the map. Due to the amount of processing involved with larger intervals, there is a preset maximum range in order to avoid stalls. The user is allowed to override this range by either disabling it completely or setting a custom value for it. Finally, it is possible to reset the colour of each ship in case the colours that have currently been randomly selected do not appropriately allow the data to be interpreted.

VI. CONCLUSIONS

In this project, we have applied data engineering techniques to process a large dataset of over 300 million entries with the goal of detecting near ship collisions. We have started with a basic algorithm based on ship size and used it along with a custom mmsi-size database to detect near-collisions in the English Channel. We have found that the safe area between 2 ships is between 4-5km and that most near-collisions happen, as expected, near ports and in the Strait of Dover, as can be seen in the visualization.

There are certainly many things one can analyze on this data: one could apply a more sophisticated algorithm of near-collision detection and/or identify specific areas or hours where/when near-collision happen more frequently; using these results and other datasets, one may correlate near-

collision events with other events, such as more fishing ships in the crab fishing season.

In hindsight, regarding the technology, one is advised to use Scala for better (theoretical) performance and library support, as Scala can easily import Java libraries which are more widely available. This is furthermore advised given the issues we have encountered with the AIS message decoder. Spark was perfectly capable of handling this large amount of data in an efficient way through the YARN scheduler on SurfSARA with the appropriate settings.

REFERENCES

[1] Allianz, *Safety & Shipping Review 2016*, 2016, http://www.agcs.allianz.com/insights/white-papers-and-case-studies/safety-and-shipping-review-2016

[2] Eric S. Raymond, *AIVDM/AIVDO protocol decoding*, Aug. 2016, http://catb.org/gpsd/AIVDM.html

[3] Zhang, W., Goerlandt, F., Montewka, J. and Kujala, P., 2015. A method for detecting possible near miss ship collisions from AIS data. *Ocean Engineering, 107*, pp.60-69.

[4] Teixeira, P., Silveira, P. and Guedes Soares, C., 2015. *Assessment of ship collision estimation methods using AIS data* (pp. 195-204). Taylor & Francis Group. London.

[5] Wang, Y., Zhang, J., Chen, X., Chu, X. and Yan, X., 2013. A spatialtemporal forensic analysis for inlandwater ship collisions using AIS data. *Safety science*, 57, pp.187-202.

[6] Silveira, P.A.M., Teixeira, A.P. and Soares, C.G., 2013. Use of AIS data to characterise marine traffic patterns and ship collision risk off the coast of Portugal. *The Journal of Navigation*, 66(6), p.879.

[7] International Maritime Organization (IMO), *COLREGS - International Regulations for Preventing Collisions at Sea*, 1972, http://www.mar.ist.utl.pt/mventura/Projecto-Navios-I/IMO-Conventions%20(copies)/COLREG-1972.pdf