AIS Compression and Vessel Movement Analysis During Economic Shifts

Andrew Mueller
amuell11@vols.utk.edu

Peyton Moore
pmoore34@vols.utk.edu

***Abstract*—Major events have a strong impact on the traffic flow of multiple different types of vessels. The following paper is twofold. The first focus of this paper is finding a way to compress the dataset. Due to the sheer size of the dataset we utilize, we write a compression algorithm that demonstrates enormous space savings with the data. We aim to have a lossless compression of the data. The second focus of this paper is analyzing the effect COVID-19 had on vessel traffic in the year 2020 and how it differs from the previous year, 2019. The dataset used contains relevant vessel data, such as identification numbers, time codes, and type and position of the vessel, that were used to analyze the differences and patterns between these two years.**

***Index Terms*—MMSI, UTM Zone, SOG, COG**

I. Introduction

U.S. ports receive a constant flow of traffic, which includes cargo ships, tankers, passenger ships, and pleasure craft [1]. On average, around 60,000 vessels will arrive at a U.S. port annually [2]. Vessel traffic flow can be interrupted by numerous different sources, causing delays and increased costs. With the amount of vessels that go through the U.S., these vessels must arrive promptly. The most common factors that can cause these disruptions are pandemics, weather-related events, and some changes in trade policy. The 3 U.S. coasts, the West, East, and Gulf, are the main points of interest that need to be examined when analyzing the effects of the events on vessel traffic. The main focus for this project is going to be the West Coast and the effect COVID-19 had on the flow of traffic coming out of its ports during 2020 [3]. The West Coast, mainly the ports of Los Angeles and Long Beach, handles most of the Asian trade import/export, which was primarily affected by the closures and restrictions caused by the COVID-19 pandemic. This project will analyze data from 2019 and compare it to the data obtained from 2020 to see how great the difference was between those two years.

The second aim of this paper is to introduce a lossless compression technique in order to drastically decrease the size of the data in which we are dealing with. Our results of the algorithm allow for approximately 5.7 times for files in the same space.

The following will present our compression technique and the data analysis for ship movement on the West Coast of the United States.

II. Dataset

The following section will introduce our dataset. We utilize broadcast points from the Marine Cadastre vessel traffic dataset. The dataset starts from 2009 and is broken down by day. The data is stored and distributed in comma-separated value (CSV) format, filtered to one minute, and formatted in zipped, daily files for all U.S. coastal waters. Each data point has 17 fields. Figure 1 shows each data point in each file.

	Name	Description	Example	Units	Resolution	Type	Size
1	MMSI	Maritime Mobile Service Identity value	477220100			Text	9
2	BaseDateTime	Full UTC date and time	2017-02-01T20:05:07		YYYY-MM-DD:HH-MM-SS	DateTime	
3	LAT	Latitude	42.35137	decimal degrees	XX.XXXXX	Double	8
4	LON	Longitude	-71.04182	decimal degrees	XXX.XXXXX	Double	8
5	SOG	Speed Over Ground	5.9	knots	XXX.X	Float	4
6	COG	Course Over Ground	47.5	degrees	XXX.X	Float	4
7	Heading	True heading angle	45.1	degrees	XXX.X	Float	4
8	VesselName	Name as shown on the station radio license	OOCL Malaysia			Text	32
9	IMO	International Maritime Organization Vessel number	IMO9627980			Text	7
10	CallSign	Call sign as assigned by FCC	VRME7			Text	8
11	VesselType	Vessel type as defined in NAIS specifications	70			Integer	short
12	Status	Navigation status as defined by the COLREGS	3			Integer	short
13	Length	Length of vessel (see NAIS specifications)	71.0	meters	XXX.X	Float	4
14	Width	Width of vessel (see NAIS specifications)	12.0	meters	XXX.X	Float	4
15	Draft	Draft depth of vessel (see NAIS specifications)	3.5	meters	XXX.X	Float	4
16	Cargo	Cargo type (see NAIS specification and codes)	70			Text	4
17	TransceiverClass	Class of AIS transceiver	A			Text	2

Fig. 1. IS Dataset Table with Field Descriptions [5]

III. Compression algorithm

Overview

Our first goal of this project is to write a lossless compression algorithm. The following sections describe how we designed our algorithm, including key observations from the dataset assisting in designing our algorithm, results from the compression algorithm, and some outstanding problems with our algorithm.

When attempting to analyze our dataset, we encountered a major hurdle, namely, the size of each data file. On average, we found that each data file is approximately 750 MB. Because of this, we ran into a space problem that inspired us to come up with a way to compress the data so that we can fit more data on disk. In designing our algorithm, we aim to maintain data integrity by designing a lossless compression algorithm.

In designing this algorithm, we make a few assumptions about the data.

First, we assume the following byte sizes for each type we utilize; A long long will be 8 bytes, an integer will be 4 bytes, a short will be 2 bytes, and a float is 4 bytes.

Second, we assume that the only fields that will vary will be fields; (2, 3, 4, 5, 6, 7). All other fields should remain the same.

A. Key Observations from the Dataset

By analyzing the data files, we made several key observations that influenced the design of our algorithm.

First, each line in the data file represents a unique broadcast point from a specific vessel, providing information such as the vessel's location, speed, and status at a given time. The MMSI (Maritime Mobile Service Identity) field is a unique identifier for each vessel, allowing for differentiation between individual broadcasts.

Additionally, we observed that not all fields in the data set are populated for each line. This inconsistency introduces sparsity in the data, which we can exploit for compression.

Furthermore, the dataset contains a variety of data types across its fields, such as numeric values (e.g., latitude and longitude), text (e.g., vessel names and call signs), and categorical codes (e.g., vessel types and statuses). Understanding these characteristics was crucial in identifying patterns and redundancies that informed our compression strategy.

B. Approach

In writing our compression algorithm, we decided that we wanted it to be controllable and fast. Thus, we decided to write the algorithm in C++, as Python has too much over-head of memory allocation.

The main representation of the data is an unordered map, with a key-value pair of the MMSI and the rest of the vessel information condensed into a class. Within the class, every field is stored as it is from the compressed version. The biggest field we focused on is condensing fields 2- 7 within each field. These are the fields which vary with every entry per vessel from the csv file.

Compressing fields

MMSI Field: The key observation we make with the MMSI field is that it's all digits. Even though the representation is a character array, we can condense this to an int, a primitive data type.

Because the largest number you can obtain with 9 digits fits into 30 bits, we can condense this to an integer instead of its character representation.

A side effect of storing this in an integer is hashing is faster.

DateTime Field: This field has two components, the date and time. If we observe that the date does not change across each entry, we only need to store that globally for each vessel. The time is handled separately, but because we need to store that information for each entry, we make another observation that the time field is 8 characters.

Each component is 6 digits, which can fit into an integer, however, because it is 6 digits, a full integer is not necessary, but just 3 bytes. Using bit shifting, we can store each digit into 4 bytes, where the most significant digit is stored in the most significant half byte.

1) latitude and longitude: When working with both latitude and longitude fields, taking note of the resolution is key. The longitude field uses eight digits and its resolution allows precise representation in just seven significant digits. Both latitude and longitude are initially stored as doubles, however, due to their small increments and limited integer portions, we can compress them into a single unsigned integer through bit shifting.

The integer portion of both fields fits into approximately 1.5 bytes, allowing us to store these integer parts together in the upper 1.5 bytes of a four-byte integer. Because the latitude integer portion is only two digits, the most significant half-byte of this region will remain zero. Meanwhile, the fractional portions of both fields fit neatly into the lower 2.5 bytes. By combining these parts, we effectively compress both latitude and longitude into one four-byte integer.

This compression strategy not only reduces storage requirements but also enhances computational performance. Because modern CPUs are heavily optimized for integer operations, comparisons and arithmetic using integers are generally faster than their floating-point counterparts. Although longitude values are technically signed and always negative, we can treat them as unsigned during compression and then restore their signed meaning when interpreting the compressed values.

In addition to performance gains, the uniqueness of each longitude value ensures that there will never be duplicate integer representations, thus preserving one-to-one mappings from the original fields. By leveraging precise bit manipulation and careful segmentation of the integer and fractional parts, we effectively halve the memory footprint and speed up computations, delivering both efficiency and accuracy in handling latitude and longitude data.

2) SOG, COG, Heading, length, width, and draft fields: These follow the same logic we use in latitude and longitude. If we note the resolution of all the fields, we can notice we don't need an integer, but just a Short, thus saving us 2 bytes for each field. The upper 1.5 bytes will store the integer portion, while the lower half byte will store the decimal portion. This is important since the data can be millions of entries, thus cutting space down by half is important for compression, as well as the fact it is the smallest way not to lose any information since we encode each digit of these fields and can extract it out using bit shifting.

3) text fields: These are relatively simple, and can not be compressed further without losing information. We write the size of each field as a char since the size of any one field does not exceed 32 characters, allowing us to know if we can skip that field in decompression, i.e., size of zero, or allocate a size of n. We can use a char since each field is restricted to 32 chars. Putting the size first speeds up decompressing since we can allocate once, and then read in the entire text bytes.

4) status: Status can be compressed down to a char. The reason is because the range is 0 - 15, which is precisely one byte.

C. Reading in data

Listing 1. Processing vessels_map

```
unordered_map<MMSI, ship*> vessels;

for (line in file)
    f1 = line{0, 1, 8 - 16}];
    f2 = line{1 - 7};
    if (f1[0] not in vessels)
        insert <vessels[MMSI], ship>;
        vessels[MMSI]->insert(f1);
    vessels[MMSI]->insert(f2);
```

Listing 1 shows pseudo-code of how the algorithm works. Because f2 is assumed non-variant, utilizing the compression of each of these fields, we create a 2d array, where each outer listing is a broadcast location point, along with the time. Each inner element is then 6 fields long. We encode the length of this array at the beginning of the array since we know what the size is, thus speeding up decompression.

We utilize an unordered map because of the speed of look-ups compared to a map structure, O(1) vs O(log n).

MMSI	Date	Name length	Name	IMO	Callsign len	Callsign
4	3	1	l(name)	4	1	l(callsign)

Vessel Type	Status	Length	Width	Draft	Cargo len	Cargo
2	1	4	4	4	1	l(cargo)

TC len	Transciever Class	TS length	time	lat	lon	SOG	COG	Heading
1	l(transciever Class)	4	3	4	4	2	2	2

Fig. 2. Resulting byte layout for each vessel entry

Figure 2 shows the resulting byte layout for each of the compressed entries. A few notes. The last 6 fields are repeated for however many entries there are in TS, so if there are 20 broadcast points, we will write an array of 20 17 bytes for that one entry. We define l(FIELD) to be the length of the field in bytes. This applies only to text fields.

This is a simple but effective approach as we will observe in the following section.

D. Results

In this section, we will look at a few important metrics. We analyze the effectiveness of our approach through two key metrics, compression ratio, and speed up.

Statistic	Speed up	Space savings %	Compression ratio
Average	13.65	82.63	5.76
Min	10.96	82.56	5.733
Max	26.09	82.72	5.79

Fig. 3. Min, Max, Average

As we can see in Figure 4, we drastically reduced the number of bytes of the raw file, and the compressed file. These raw files are on about the order of 720 MB, versus the compressed format which is about 120 MB, a compression ratio of on average 5.76%. This would allow approx 5.7 times more files in the same space, which is very useful if you are

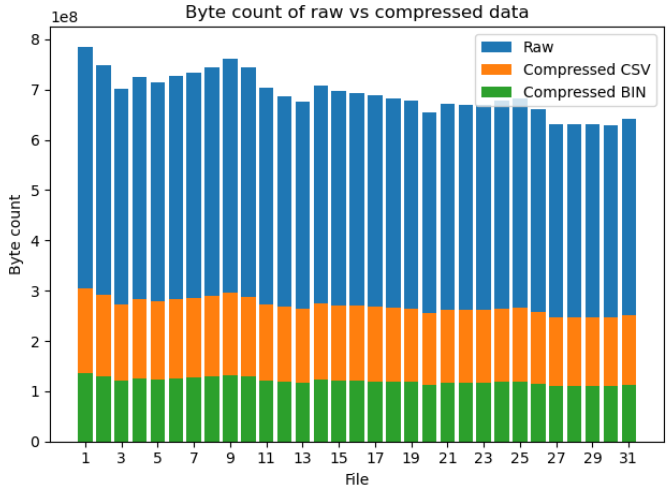

Fig. 4. Size reduction of csv vs compressed for 2019/01/01

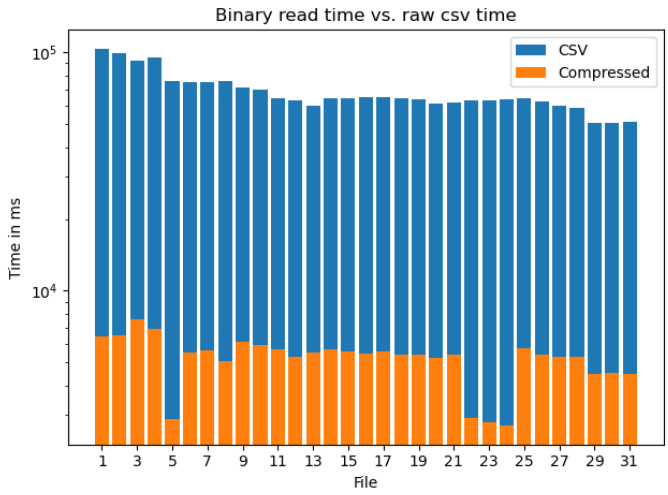

Fig. 5. Speed of reading csv vs compressed for 2019/01/01

dealing with low storage, as well as large systems that want to reduce the use of the file system. This is further shown by the average space savings, which is $\tilde{8}0\%$ per file.

Additionally, we wanted to see how the compression algorithm works in a CSV file format. Although we were not able to find read times in this compressed CSV format, we were able to get the byte count of each which is shown in fig 4. This yielded an average compression ratio for the month of January of 2.56. Although it is not as good as the binary representation, it is still useful if you want to be able to read in the CSV format.

Because we are working at the byte level, it is expected to be faster to read since we don't have to worry about conversion, and essentially just copy memory which is usually highly optimized. As we can see from figure 5, we have significant gains in the speed in which we can read binary data versus the raw data. We can see some anomalies, which have to be

studied further, but on average we can get a speed up of approx 13.65%. This is great if a user has loads of files to load up.

E. Conclusion

Given our enormous dataset, we aim to have a loss less compression algorithm. Using key insights into the data, and how it is formatted, we were able to design a compression algorithm that is centered on understanding byte level information. We are able to see that our compression algorithm achieves approximately a 5.7 compression ratio, with an average speed up of reading in the compressed binary files of 13x faster than reading in the raw CSV format.

Because of the nature of our algorithm, most of the data types can be converted into integer types. This allows for fater computation on CPU's since integer math is highly optimized on most modern CPU's. This is great because we want to be able to perform fast analysis, which integer types aid in doing.

F. Future work

We would like to do a few things in the future

First, we would like to see how map-reduce affects the compression time, and decompression times

Second, we would like to introduce some more metadata into the compression in hopes we can see an increase in speed up of reading in the files, as well as allowing support for multithreading.

Third, we would like to see further how much information is lost if any. A specific field in which might change across broadcast points is the vessel status. If this is the case, we would need to find a way to encode that. I believe that if it does change, we can utilize a unwritten half byte in our compressed data already, we can tie in any one broadcast point to a specific status.

Lastly, we would like to run this through the entire dataset to see just how much data can be compressed.

IV. COVID-19 ANALYSIS

...

REFERENCES

[1] "Vessel Traffic - Marine Cadastre," Marinecadastre.gov, 2016. https://hub.marinecadastre.gov/pages/vesseltraffic (accessed Nov. 21, 2024).

[2] M. Chambers and M. Liu, "Maritime Trade and Transportation by the Numbers — Bureau of Transportation Statistics," Bts.gov, 2017. https://www.bts.gov/archive/publications/by_the_numbers/maritime_trade_and_transportation/index

[3] U.S. International Trade Commission, "The Impact of the COVID-19 Pandemic on Freight Transportation Services and U.S. Merchandise Imports — USITC," www.usitc.gov, 2020. https://www.usitc.gov/research_and_analysis/tradeshifts/2020/special_topic.html

[4] Marine Cadastre, "Automatic Identification System (AIS) Data," Available at: https://marinecadastre.gov/accessais/, Accessed: November 28, 2024.

[5] Marine Cadastre, Data dictionary, Available at: https://coast.noaa.gov/data/marinecadastre/ais/data-dictionary.pdf