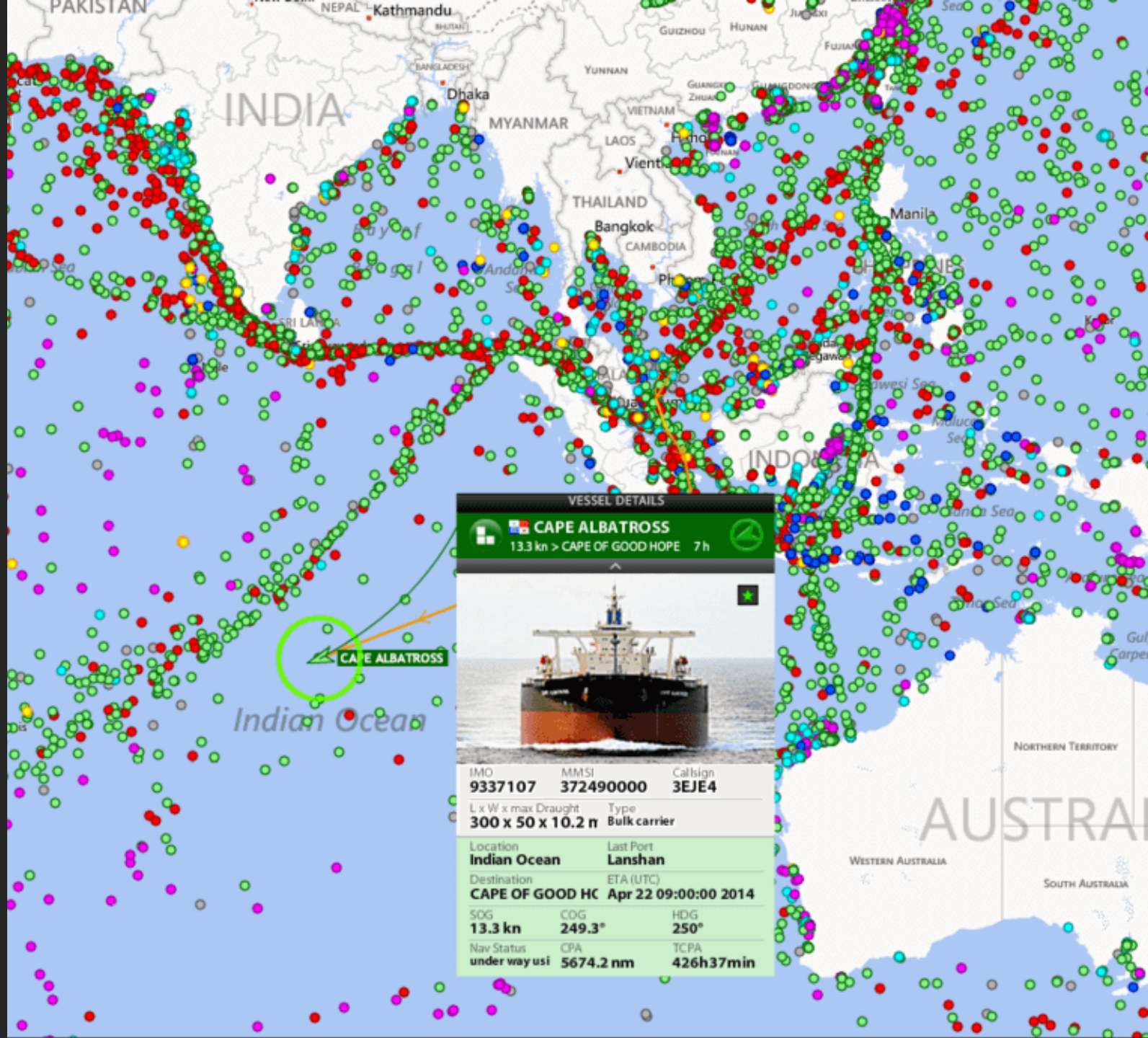
Tracking Trips: Segmenting and Analyzing GPS Trajectories From Cargo Ship Data

Outline

- Background and Problem Statement
- Motivations and Objectives
- Project Scope
- Literature Review
- Methodology
- Results & Analysis
- Conclusions
- Demo of Tool (Time Allowing)

Background and Problem Statement

- The explosion of internet-enabled devices that record location and time offer opportunities to better understand our environment.
- However, the sheer volume of this data confounds many traditional geospatial information systems and methods. Furthermore, specialized approaches developed in the last 15 years are difficult to understand and use.
- To explore this problem set, we will leverage Automatic Identification System (AIS) ship tracking data.

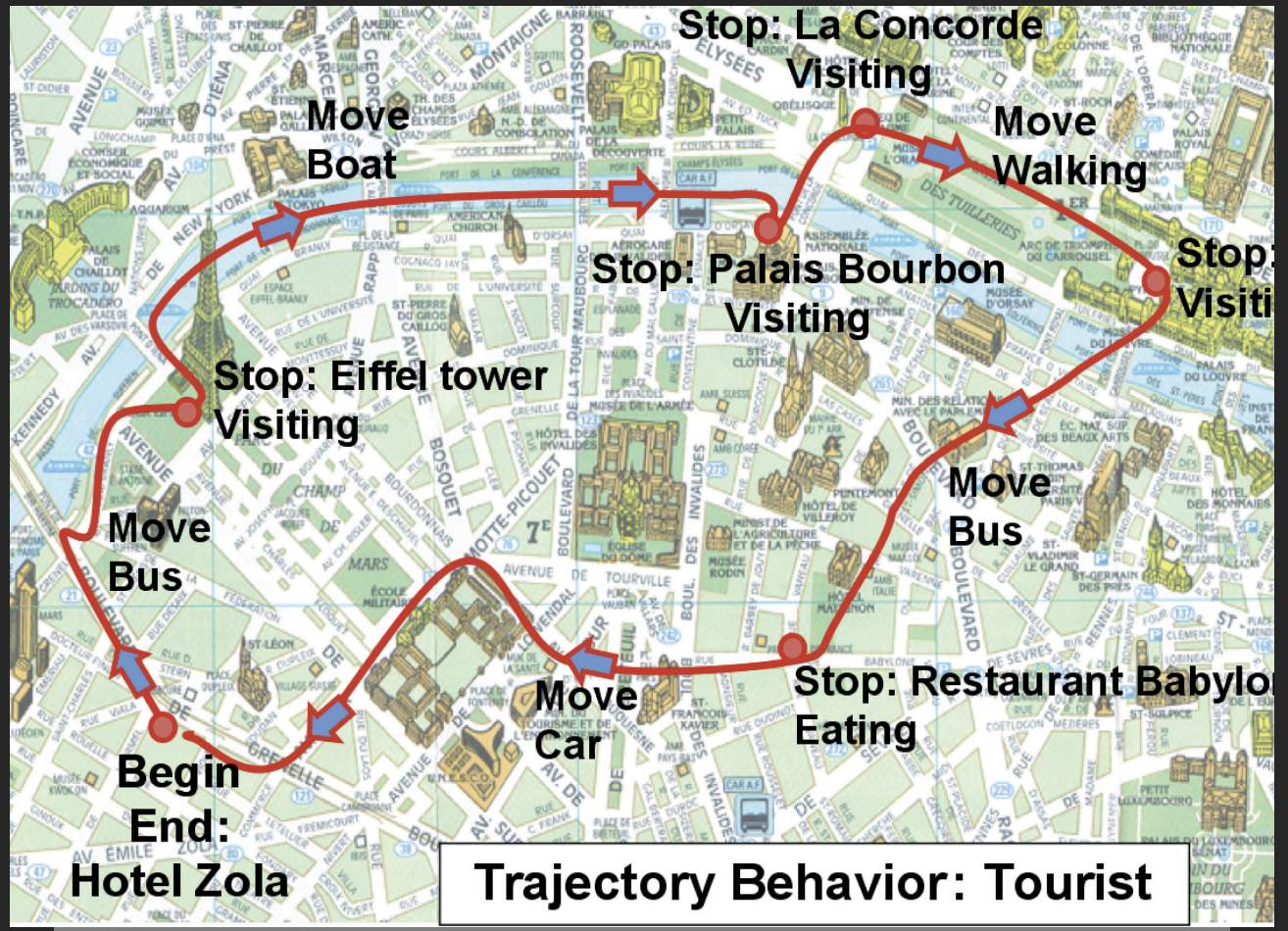


Motivation & Objectives

Frequently, data scientists need to identify which users went where, when they arrived, and when they left. Today's available methods to approach this process—known as trajectory segmentation—are complex and can be difficult to apply in production environments.

In this project, I will pursued three objectives:

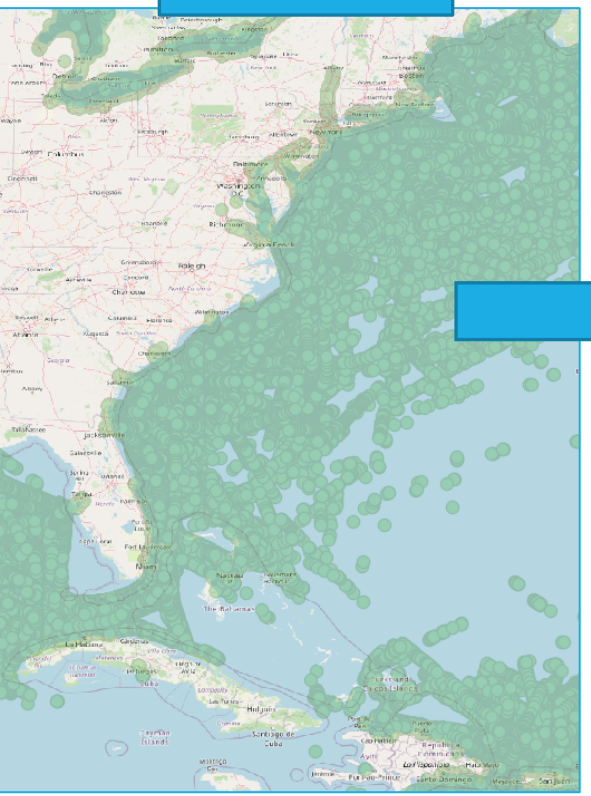
1. Identified a **subset of scalable approaches** to parse trajectories
2. **Developed (or employed already-developed) implementations** approaches using Python and PostGreSQL.
3. **Evaluated and optimized** these implementations in a production-like environment with millions of points.



The notional trajectory of a tourist visiting Paris to highlight trajectories, stops, and moves. From "Semantic Trajectories Modeling and Analysis," 2013, Spaccapietra et all

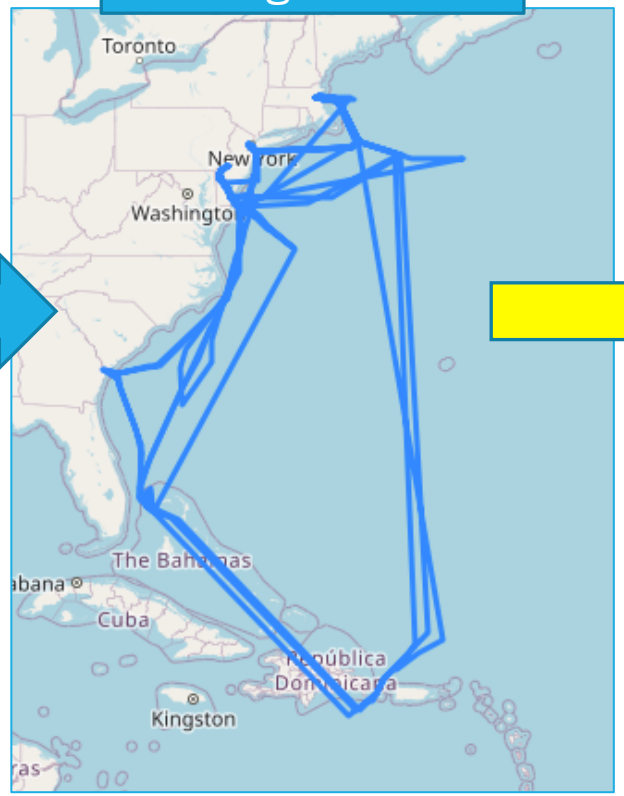
Project Scope

Raw Data



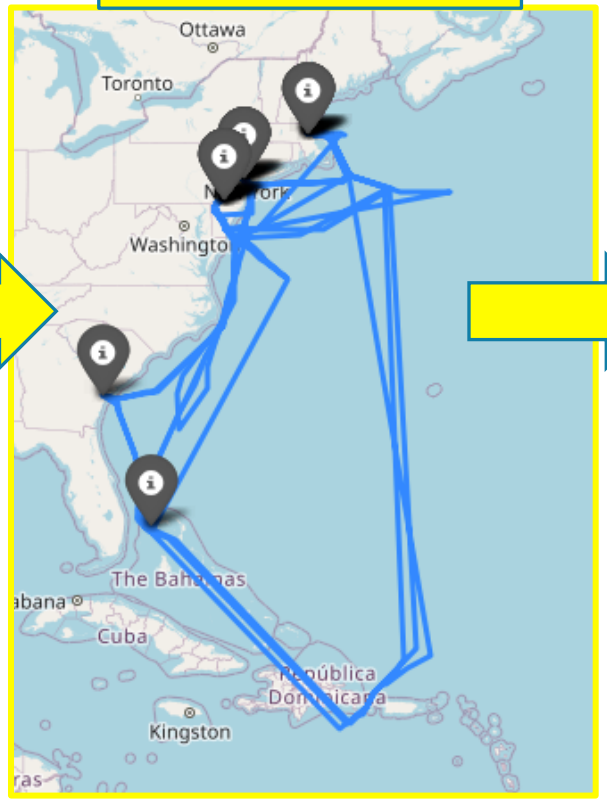
This data shows how difficult it can be to make sense of a large geospatial dataset plotted in its raw format. Manually identifying where ships are going is complex and time intensive.

UID Segmentation



Segmenting data by unique ID (Maritime Mobile Service Identity or MMSI for ships), ordering by time, and visualizing the data shows the path of a ship and suggests likely port visits.

Site Identification



Our goal is to develop an automatic segmentation process to further subdivide each individual ship's trajectory into know port visits. Priorities include ease of implementation, hyperparameter complexity, and accuracy.

Trajectory Extraction

Site Name	Arrival	Depart	Time Difference
BOSTON	2017-01-06 12:11:00	2017-01-07 07:25:00	0 days 19:14:00
NEWARK	2017-01-08 11:23:00	2017-01-09 07:05:00	0 days 19:42:00
GLOUCESTER	2017-01-10 01:31:00	2017-01-10 15:20:00	0 days 13:49:00
FREEPORT	2017-01-13 10:06:00	2017-01-14 21:51:00	1 days 11:45:00
BOSTON	2017-02-21 10:57:00	2017-02-21 21:28:00	0 days 10:31:00
NEWARK	2017-02-23 04:47:00	2017-02-23 23:58:00	0 days 19:11:00
GLOUCESTER	2017-02-24 19:44:00	2017-02-25 09:57:00	0 days 14:13:00

The desired output is a list of sites visited with arrival time and departure, which can feed additional analysis pipelines for route prediction, timeseries analysis, or clustering of ships.

Literature Review: Selected Research

Title	Description	Date	Cite
A Conceptual View On Trajectories	Foundational paper laying out terminology still used today.	2007	Source
Understanding Human Mobility Patterns	Overview of Major Trends in Human mobility patterns in Nature	2008	Source
A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise	Major foundational density-based clustering approach.	1996	Source
OPTICS: Ordering Points To Identify the Clustering Structure	Significant improvements to DBSCAN. No need for specific epsilon parameter.	1999	Source
Density-Based Clustering Based on Hierarchical Density Estimates	The development of HDBSCAN. Further reduction for need to optimize epsilon.	2013	Source
T-DBSCAN: A Spatiotemporal Density Clustering For GPS Trajectory Segmentation	One of many different methods to segment trips as trajectories.	2014	Source
ST-DBSCAN: An algorithm for clustering spatial–temporal data	Application of DBSCAN with a moving, temporal filter.	2006	Source
Scikit-mobility: A Python Library for the Analysis, Generation and Risk Assessment of Mobility Data	Development of an efficient and effective Python package including several methods for parsing trajectories.	2019	Source

Methodology: Identify Scalable Approaches

Density-Based Clustering Methods

- Clusters can occur anywhere.
- Depending on the method, can include various hyper parameters, which must be set appropriately.
- Includes DBSCAN (***) OPTICS (**), HDBSCAN (**), and ST-DBSCAN (*).
- Can include variants such as double-clustering or filtering to points with low speeds.

Static Trip Segmentation

- Clusters can only occur at known sites. Eliminates false positives but requires a robust baseline knowledge.
- Requires points to be within a maximum distance from a known site for a minimum time.
- Implemented in a previous project to translate geospatial datasets into networks (*).

Dynamic Trip Segmentation

- Clusters can occur anywhere
- Requires points to be within a maximum distance of each other for a minimum time.
- Scikit-Mobility's (SKMOB) implementation has additional features, such as filtering by minimum speed and maximum gaps in time and a fast implementation(**).

Since static trip segmentation is the most conservative method, we will use this as our "ground truth" to evaluate the effectiveness of additional methods.

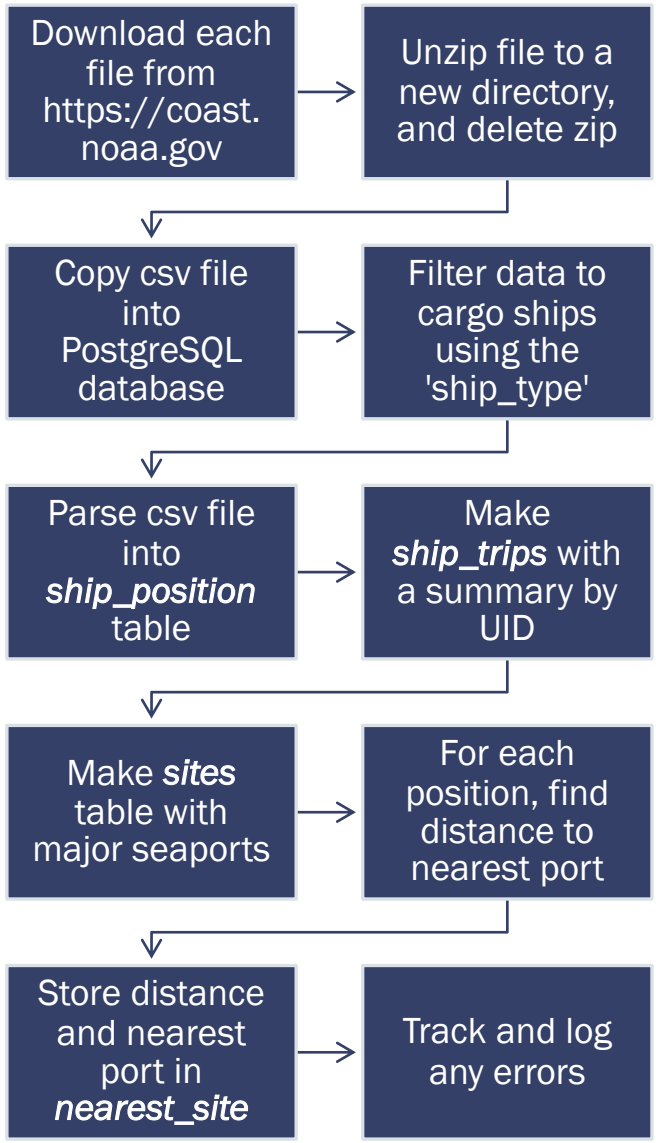
For each method, we will evaluate the number of clusters near "ground-truth" clusters and determine the **precision**, **recall**, and **F1 measure** across a range of hyper parameters.

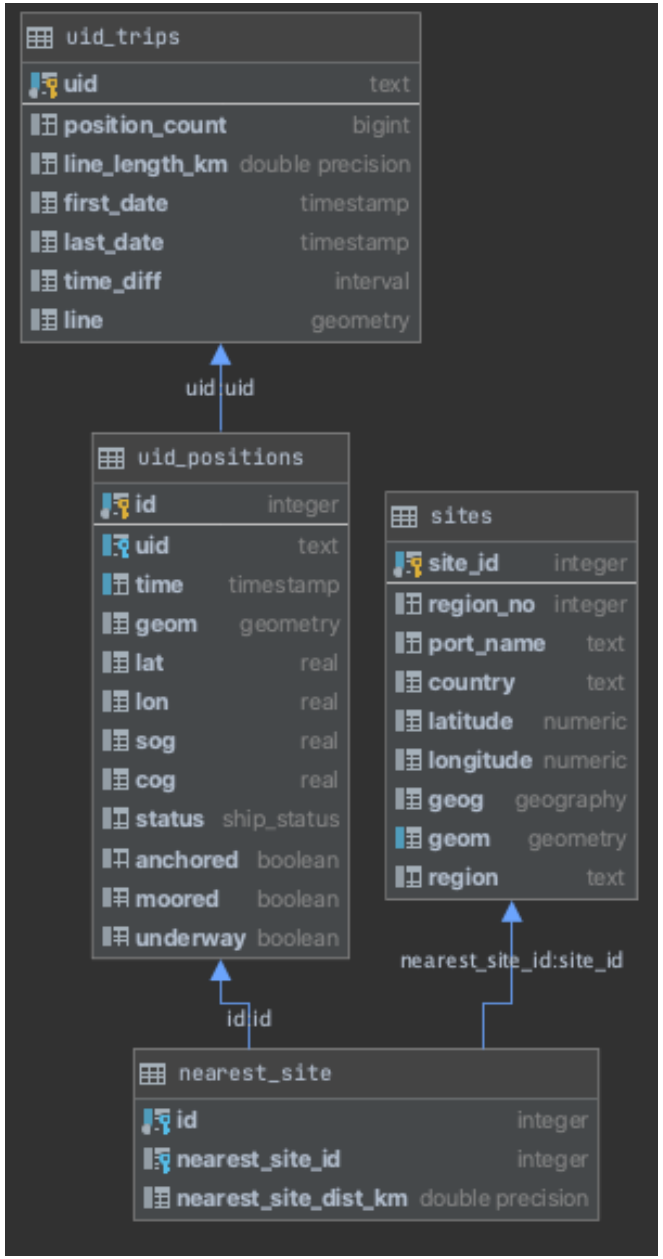
***: Numerous, optimized implementations available. **: At lease one Python implementation capable of scaling. *: No implementations or inefficient scaling.

Methodology: Develop/Employ Implementations

I scraped, processed, and parsed 250 million ship location reports using publicly available NOAA data, storing the data in a PostGreSQL database. I only used ships that were identified as cargo ships to ensure a more consistent dataset (no fishing vessels, private ships, tugs or other similar working craft).

The core tables are included in the Entity Relationship Diagram (ERD) to the right, showing the relationship between sites (ports), ships' positions, summaries of each ships' trip, and the nearest site to each position.

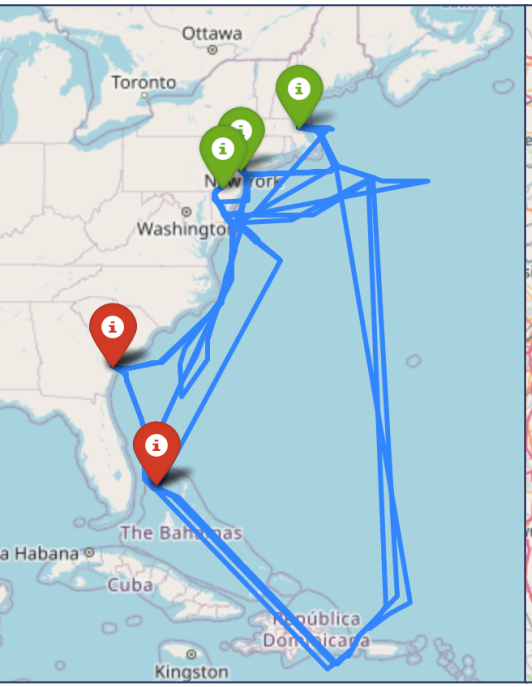




Over **200 million** geolocations in 2017, **16 million cargo ship positions from ~3,000 ships** in January. Total Database size of 71 gigabytes

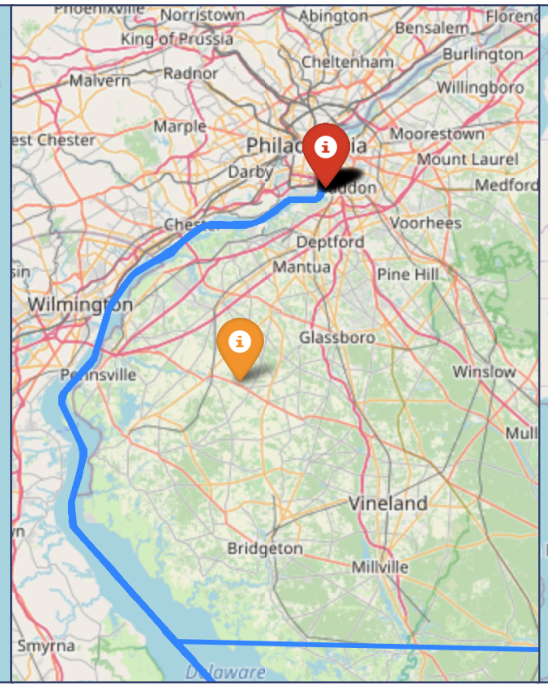
Methodology: Evaluate and Optimize Approaches

To explore and compare how the algorithm implementations worked and to ensure the accuracy of my metrics, I developed a Python package and built a Jupyter Notebook tool to plot and analyze different clustering results.
Green Points are True Positives, Red Points are False Negatives, and Orange Points are False Positives.



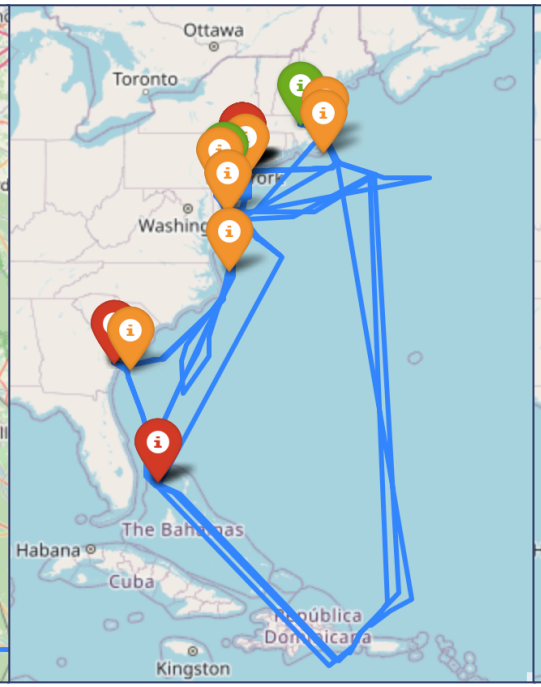
DBSCAN with high parameters. Only cluster activity is at three major ports.

Precision: 1.0 Recall: .10 F1: .18



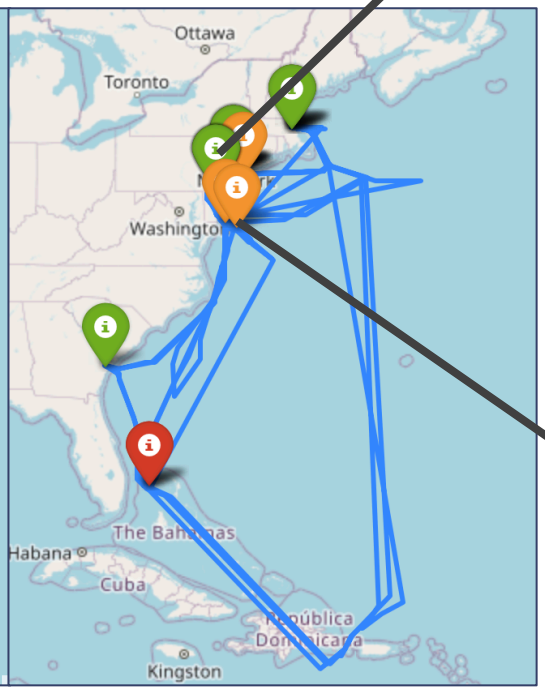
DBSCAN with low parameters. Activity far from port is included in cluster and moves cluster center.

Precision: .21 Recall: .10 F1: .13



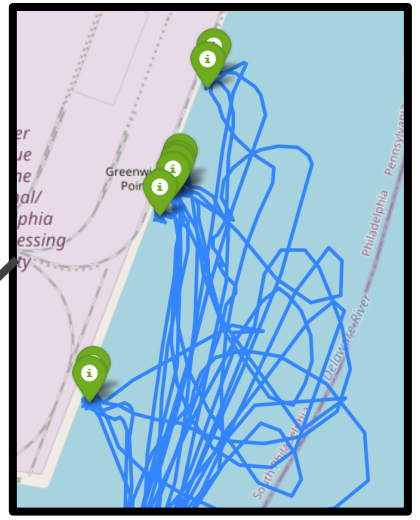
HDBSCAN with too low parameters. Numerous false positives and negatives at sea and near ports.

Precision: .21 Recall: .10 F1: .13

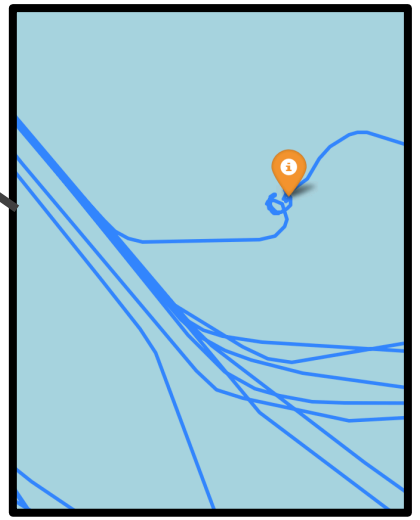


OPTICS with tuned parameters. Still misses, but numerous correct clusters.

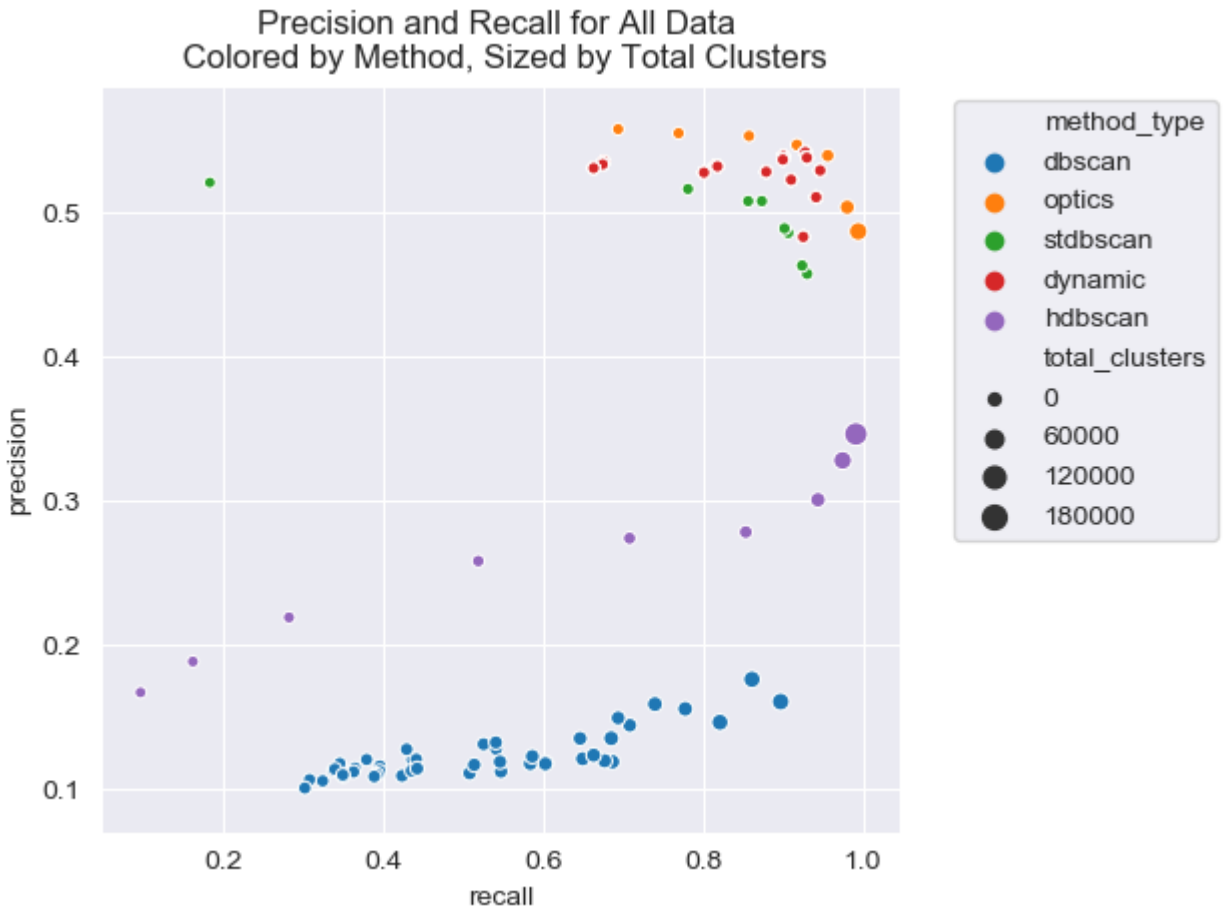
Precision: .88 Recall: .71 F1: .79

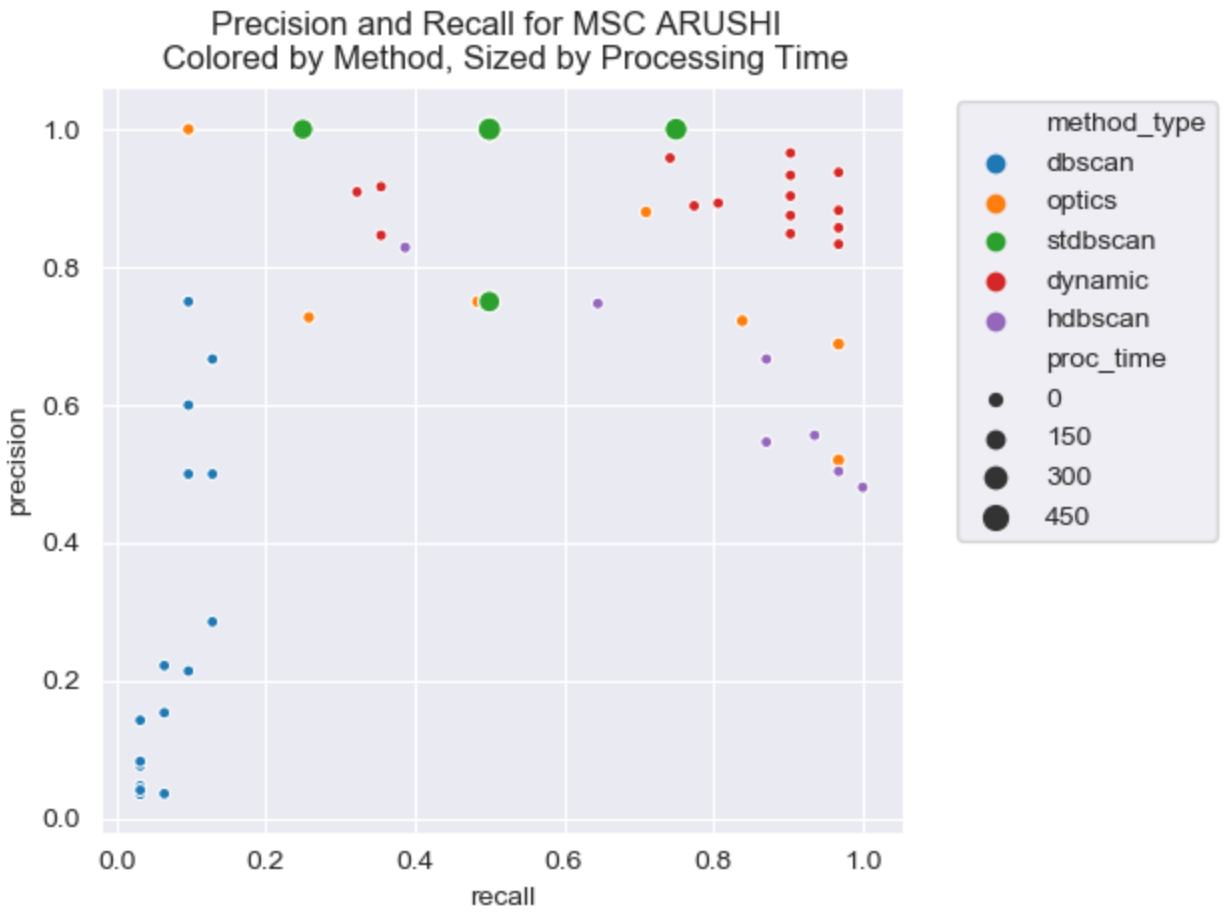


Even without a time filter, OPTICS successfully clusters each port visit in space.



Some false positives are likely anchorages and could be considered true “stops.”





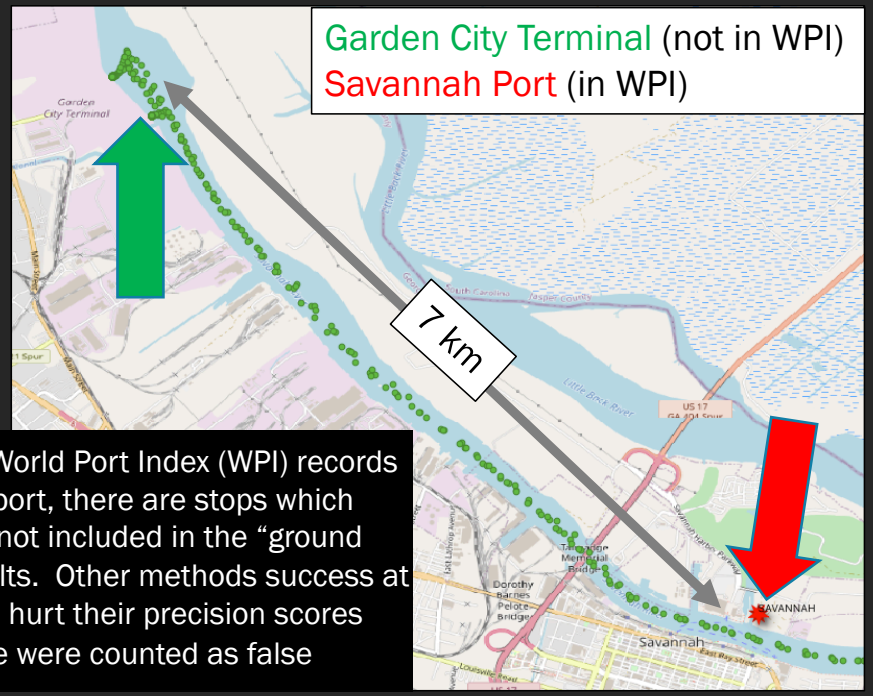
Results and Analysis
Final Metrics

SCORES WERE NOTABLY HIGHER FOR THE SAMPLE SHIP ANALYSIS, LIKELY BECAUSE A CLEAR "EASY" EXAMPLE WAS SELECTED.

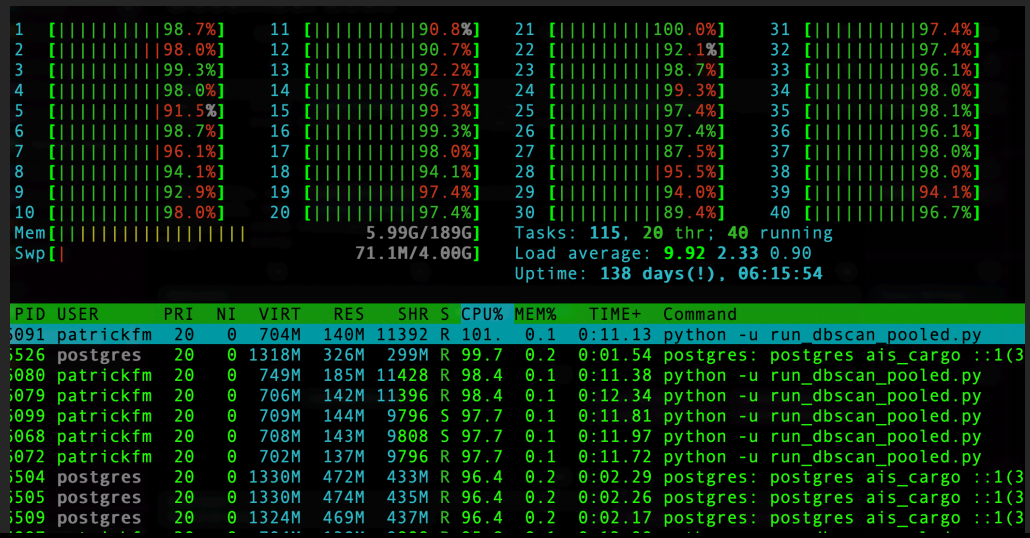
OPTICS AND DYNAMIC SEGMENTATION PERFORMED THE BEST OUT OF ALL THE METHODS.

Conclusions

- The three best performing models across the 16 million records are dynamic segmentation, OPTICS, and ST-DBSCAN.
  - Dynamic Segmentation performed well, but requires hyperparameters that may need subject matter expertise or grid searching.
  - OPTICS requires only a minimum sample and a maximum epsilon distance, making it easier to use.
  - ST-DBSCAN took much longer (400x DBSCAN, 10x OPTICS)
- Additional deep dives into other ships or using a better site list could improve overall performance. The sample ship data performed better than all ships after minimal data conditioning.
- Working on a dataset with hundreds of millions of rows requires informed foresight and strict adherence to a plan.



Since the World Port Index (WPI) records only large port, there are stops which were then not included in the "ground truth" results. Other methods success at these sites hurt their precision scores since these were counted as false positives.



Clustering 16 million points with multiple algorithms across a wide hyper parameter space was a significant compute requirement. I ran a 28-core server on Colonial One nearly continuously for about a month while optimizing.

Current Limitations and Future Research

- Try other Datasets
  - Planes, Trains, and Automobiles?
- Evaluate accuracy of extracted sequence of port visits through Sequence Alignment, Mutual Information, or other metrics.
- Identify likely new “sites” that aren’t in the reference data but are frequently visited by ships.
- Include time in cluster evaluation metrics.

References

- Unacast's Social Distance Scoreboard uses geospatial data from personal cellphones to evaluate how well different regions adhere to recommended social distancing guidelines during the COVID-19 pandemic. Accessed 12 October 2020, https://www.unacast.com/covid19/social-distancing-scoreboard?view=state&fips=11
- MarineCadastre.gov's AIS data repository. Accessed on 1 September 2020, https://marinecadastre.gov/ais/
- GitHub repositories with implementations of clustering algorithms, such as https://github.com/eren-ck/st_dbscan, and https://github.com/choyixi/STDBSCAN. Accessed on 10 October 2020
- Sci-Kit Learn's Clustering Metrics, Accessed on 10 October 2010, https://scikit-learn.org/stable/modules/clustering.html


Questions?

Backup

Methodology: Tools Used

PostGreSQL 12 with Post-GIS extension

Local installation on laptop
Colonial One Server

PG Admin 4

Server management and monitoring
Testing and developing complex SQL queries

QGIS Mapping and Plotting Tool

Plotting clusters to evaluate accuracy
Directly connected to PostGres servers

PyCharm IDE with Python 3.7

Develop, test, and implement analysis
Generate Entity Relation Diagram (ERD)

Python Packages

Pandas, Numpy, SQLAlchemy, Psycopg2
Matplotlib, Folium, Geopy, GNACT
Scikit-Learn, Scikit-Mobility, HDBSCAN.

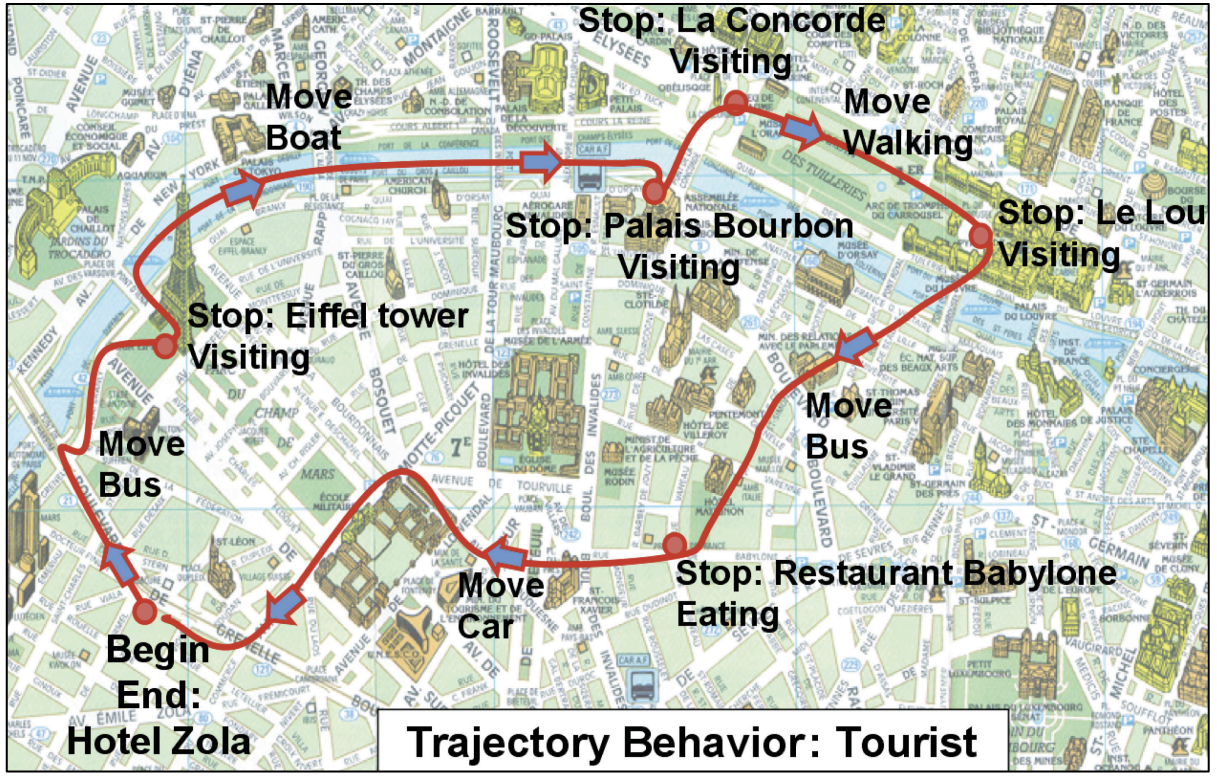
Literature Review: Key Definitions

Trajectory: A trajectory is the evolution of the position (perceived as a point) of an object that is moving in space during a given time interval.

Stop: A stop is a part of a trajectory, such that: 1) The use-case supports representing this part of the trajectory as a stop, 2) The temporal extent is a non-empty time interval, and 3) the traveling object does not move. All stops are temporally disjoint.

Move: A move is a part of a trajectory, such that: 1) The part is delimited by two extremities that represent either two consecutive stops, when time begins and a stop, or the last stop and when time ends, 2) the temporal extent is a non-empty time interval, and 3) the spatial range of the trajectory interval is along the spatio-temporal line defined by the trajectory function

UID: Unique ID that represents the user, vehicle, or piece of equipment associated with the GPS track.



The notional trajectory of a tourist visiting Paris to highlight trajectories, stops, and moves. From "Semantic Trajectories Modeling and Analysis," 2013, Spaccapietra et all

Results and Analysis: Relevant Approaches

Method	Summary	Parameters	Advantages	Limitations	Example with AIS Data
DBSCAN (Density Based Spatial Clustering of Applications with Noise)	The original and most well-known approach, Density Based Spatial Clustering of Applications with Noise (DBSCAN) finds clusters with arbitrary shape that satisfy the minimum parameters.	Epsilon (eps): minimum distance between core points Minimum Sample: Minimum number of core points for a cluster	Straightforward, easy to explain, fast processing	Without a time window, the density is relative to the time horizon of the data. Also, locations visited frequently will be indetified as stops regardless how much time is spent.	Global patterns across all UIDs. This method would find areas where ships frequently travel including major shipping lanes and ports.
Density Clustering by UID	Run DBSCAN against each ship individually, rather than globally.	Eps, Minimum Sample, and Unique ID: The unique descriptor for the user, vehicle or device producing the GPS track.	Focuses on each individual UID	Requires a structured approach to pull and order each UID's positions from the database	Find denser areas where this specific ship spent large amounts of time.
Density Based Clustering with Speed Filtering	First filter out all points faster than a certain speed, then apply DBSCAN by UID.	Epsilon, Minimum Sample, UID, and minimum speed	Faster processing times since there are fewer points. Unlikely to get expanding clusters.	Focuses on velocity, so slower speed areas like traffic jams can trigger false alarms.	Finds ports where ships are moving slower or stopped.
Density Based Clustering with Time Window (ST-DBSCAN)	Using a time window as well as a distance window for cluster formation.	Epsilon, Minimum Samples, UID, Epsilon2 (eps2): A second value for epsilon used to define a time distance threshold	Allows a lower minimum sample size without exploding cluster size.	Additional hyperparameter to optimize, if we set eps2 too high, we miss activity of interest.	If we know how long the minimum time it takes for a cargo ship to offload its cargo, we can use that as eps2.
OPTICS (Ordering Points To Identify the Clustering Structure)	Uses a reachability plot to determine tailored epsilon distances (within a specified maximum) for each cluster.	Maximum epsilon, minimum sample	Much more refined clusters with fewer parameters.	Additional time complexity for processing.	Can parse port visits with much greater detail than DBSCAN.
HDBSCAN (Hierarchical DBSCAN)	Creates a spanning tree to group similar density clusters that are near each other.	Minimum sample	Very fast	Poorly identifies the breaks between port visits.	Does a poor job with slow moving ships.
Dynamic Trip Segmentation	Uses a maximum distance threshold and minimum time threshold to segment a velocity into individual moves.	Max distance and min time required for a cluster to form.	Much faster and easy to explain.	Applies a global parameter for the entire trajectory, rather than a density based estimate.	Any time there are points within 3km of each other for more than 6 hours, create a cluster.
Static Trip Segmentation	Uses a maximum distance threshold, minimum time threshold, and list of known sites.	Max distance and min time required for a cluster to form, list of sites that will be stops.	Very low or non-existent false positive rate.	Requires a list of known stop sites, and will only find stops at those sites.	Given a list of known ports, create a cluster anytime there are points within 3km of a known port for more than 6 hours.

Walking through an example…

The container ship MSC ARUSHI R.

We can plot the positions in QGIS showing where the ship has been. Note the lack of activity far from the continental US or Puerto Rican coast lines where there is no coverage.

The Mediterranean Shipping Company's cargo ship MSC ARUSHI R. was present in the 2017 AIS data 40,614 times.

MSC ARUSHI R. (IMO: 9244881, (MMSI: 636016432) is a container ship that was built in 2002 and is sailing under the flag of Liberia.

Its carrying capacity is 4112 twenty-foot equivalent unit cargo containers and her current draught is reported to be 10.6 meters. Her length overall (LOA) is 281 meters and her width is 32.2 meters.

The ship was active from January to December 2017 in the data and traveled about 38,000 km.

ETL and Data Ingest: MSC ARUSHI R.

After the data is downloaded from the website and unzipped, the csv file is read into the database where the raw data is parsed. If the MMSI is described as a cargo ship in the “ship_type” field, a unique id is assigned and the MMSI, time, latitude, and longitude of the position is written to a new table with all other cargo ship positions. A new field called “geog” is also created, which is a special data type used by the database.

	id integer	mmsi text	time timestamp without time zone	geog geography	lat numeric	lon numeric
1	13266793	636016...	2017-01-06 07:04:31	0101000020E61000...	42.28535	-69.12919
2	13266794	636016...	2017-01-06 07:06:31	0101000020E61000...	42.28576	-69.14447
3	13267112	636016...	2017-01-06 07:07:55	0101000020E61000...	42.28601	-69.15519
4	13267111	636016...	2017-01-06 07:09:08	0101000020E61000...	42.28627	-69.16453
5	13266799	636016...	2017-01-06 07:10:13	0101000020E61000...	42.28669	-69.17284

	mmsi text	ship_name text	ship_type text
1	636016432	MSC ARUSHI R	1004

Details about the ship from the raw data are written to a different table, ship_info, for each MMSI.

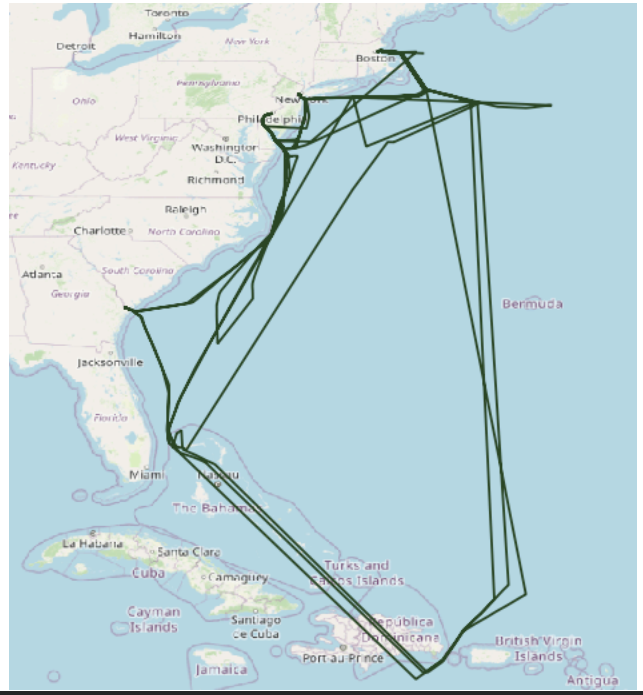
ETL and Data Ingest: MSC ARUSHI R.

	mmsi text	position_count bigint	line_length_km double precision	first_date timestamp without time zone	last_date timestamp without time zone	time_diff interval	line geometry
1	636016432	40614	37839.07705355832	2017-01-06 07:04:31	2017-12-31 23:59:14	359 days 16:54:43	...

A final summary table is created with a count of all positions, first and last date detected, the time difference between these two dates, and the shape of the line that connects all the points ordered by time.

By using PostGres' built-in GIS capabilities, we can easily make the 40,000 points into one line segment.

This allows for easier analysis of ships' journeys, particularly where there is no coverage in the data.



Nearest Port Calculation: MSC ARUSHI R.

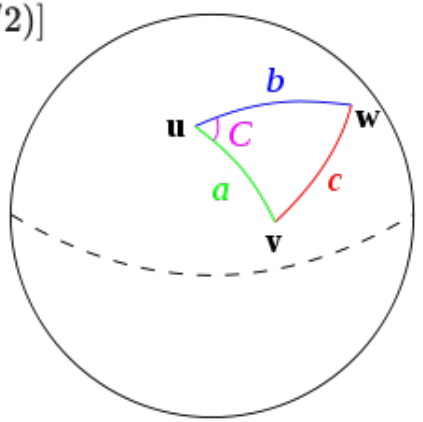
For each point, we can now determine what the nearest port is from our list of known ports and the distance to it. To save space, we use the Port ID in our database. In this example, 7290 corresponds to Provincetown, a port on the end of Cape Cod in Massachusetts.

By spending the resources now to calculate the distance to the nearest port for all positions, we can set filters further down the production stream to identify the threshold for considering a ship is “in-port”.

	id integer	mmsi text	time timestamp without time zone	nearest_port_id integer	nearest_port_dist_km double precision
1	13266793	636016432	2017-01-06 07:04:31	7290	90.73337530450199
2	13266794	636016432	2017-01-06 07:06:31	7290	89.54141052460032
3	13267112	636016432	2017-01-06 07:07:55	7290	88.70508209894432
4	13267111	636016432	2017-01-06 07:09:08	7290	87.97855285944556
5	13266799	636016432	2017-01-06 07:10:13	7290	87.33898039939933

$$D(x, y) = 2 \arcsin[\sqrt{\sin^2((x1 - y1)/2) + \cos(x1)\cos(y1)\sin^2((x2 - y2)/2)}]$$

Distance between two points on the surface of the earth is best computed using the Haversine formula (shown above), which accounts for the curve of the earth. This is comparable to the “great circle” distance. In this implementation, we used the Haversine knn function from the Python package Sklearn.

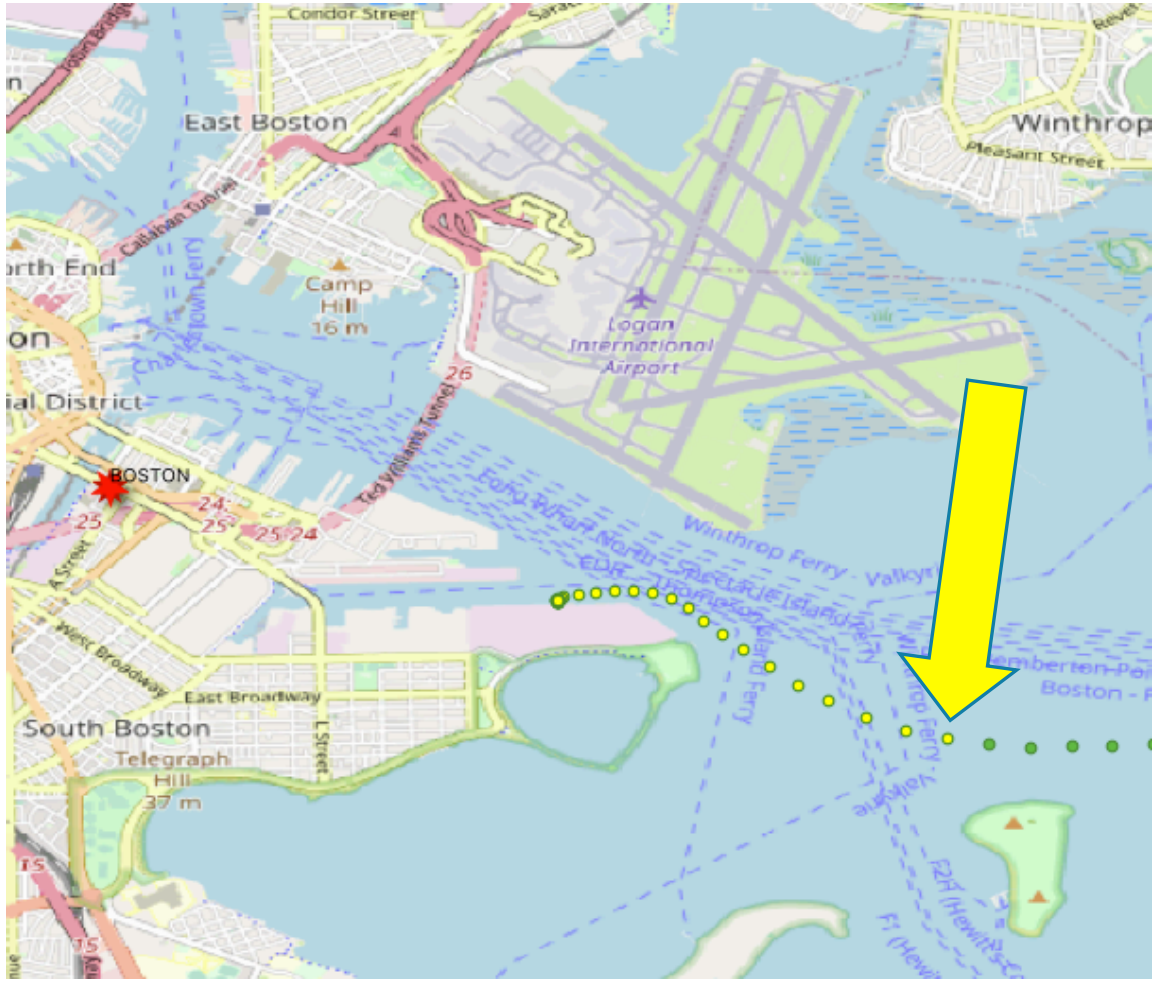


Determining Stops: MSC ARUSHI R.

1. Determine distance threshold to be considered ”in-port”. In our example, we used 5km coupled with a minimum loiter time (to be addressed later).
2. For each position, if the closest port’s distance > dist value, the “node” that the ship is at is 0 which stands for open sea or anytime a ship is greater than 5km from any port in out database. If the distance < our dist value, the node is the port id of that port.

index	time	node	next_node	prev_node
262	2017-01-06 12:01:08	0	0.0	0.0
263	2017-01-06 12:02:18	0	0.0	0.0
264	2017-01-06 12:03:28	0	7250.0	0.0
265	2017-01-06 12:04:37	7250	7250.0	0.0
266	2017-01-06 12:05:47	7250	7250.0	7250.0
267	2017-01-06 12:06:58	7250	7250.0	7250.0

In the table above, MSC ARUSHI R.’s 265th position is the first considered in port and is a transition from node 0.



In this map, MSC ARUSHI R.’s positions are colored yellow once the ship is within 5km of Boston Harbor. These positions count as “in-port” while the green positions are assigned to node 0.

Determining Stops: MSC ARUSHI R.

3. For each position, add a column for the previous node visited and the next node visited. Anytime the previous node does not equal the next node, there is a state change from one node to another. If previous node == next node, nothing has changed.
4. If the current node == next node, but the next node != previous node, the ship has just entered a new node. If the current node == previous node, but the next node != previous node, the ship is leaving a node. We can use this speedy comparison to determine when a ship enters or leaves a node and build a new dataframe of all stops for each ship (shown below).

The table below shows the first transition from node 0 to node 7250, including the relevant depart and arrival times.

	node integer	arrival_time timestamp without time zone	depart_time timestamp without time zone	time_diff interval	destination integer	position_count bigint	mmsi text
1	0	2017-01-06 07:04:31	2017-01-06 12:03:28	04:58:57	7250	265	636016432
2	7250	2017-01-06 12:04:37	2017-01-07 07:31:50	19:27:13	0	407	636016432
3	0	2017-01-07 07:33:10	2017-01-08 10:07:07	1 day 02:33:57	7830	1301	636016432
4	7830	2017-01-08 10:08:17	2017-01-08 10:41:57	00:33:40	7850	31	636016432
5	7850	2017-01-08 10:43:06	2017-01-08 11:20:50	00:37:44	7810	35	636016432
6	7810	2017-01-08 11:21:57	2017-01-09 07:05:57	19:44:00	7850	406	636016432
7	7850	2017-01-09 07:07:07	2017-01-09 07:55:48	00:48:41	7830	45	636016432
8	7830	2017-01-09 07:56:50	2017-01-09 08:27:27	00:30:37	0	29	636016432
9	0	2017-01-09 08:28:31	2017-01-09 22:53:35	14:25:04	8030	778	636016432
10	8030	2017-01-09 22:54:41	2017-01-09 23:13:04	00:18:23	8040	18	636016432

Source	Target	Source Depart	Target Arrival
BOSTON	NEWARK	2017-01-07 07:31:50	2017-01-08 11:21:57
NEWARK	GLOUCESTER	2017-01-09 07:05:57	2017-01-10 01:26:56
GLOUCESTER	FREEPORT	2017-01-10 15:24:50	2017-01-13 10:06:34
FREEPORT	BOSTON	2017-01-14 21:51:17	2017-02-21 10:51:03
BOSTON	NEWARK	2017-02-21 21:33:12	2017-02-23 04:47:15
NEWARK	GLOUCESTER	2017-02-23 23:58:58	2017-02-24 19:39:49
GLOUCESTER	BOSTON	2017-02-25 10:01:59	2017-04-12 08:09:08
BOSTON	NEWARK	2017-04-12 20:18:28	2017-04-14 09:51:02
NEWARK	GLOUCESTER	2017-04-15 00:32:49	2017-04-15 18:58:26
GLOUCESTER	BOSTON	2017-04-16 07:55:54	2017-06-01 09:02:25
BOSTON	FREEPORT	2017-06-29 05:04:22	2017-07-05 14:57:14
FREEPORT	BOSTON	2017-07-06 07:00:27	2017-08-08 09:35:54
BOSTON	NEWARK	2017-09-14 06:39:29	2017-09-15 06:48:31
NEWARK	GLOUCESTER	2017-09-16 07:24:56	2017-09-17 00:19:09
GLOUCESTER	BOSTON	2017-09-28 12:48:58	2017-12-27 11:35:18
BOSTON	GLOUCESTER	2017-12-27 23:16:46	2017-12-29 16:01:22
GLOUCESTER	NEWARK	2017-12-30 05:23:35	2017-12-31 01:44:40

In the table, our edgelist starts with Boston. Since it is the first node visited in the data, we can't say where the ship came from before it arrived at Boston.

Building Edgelist: MSC ARUSHI R.

From the full list of stops, we can now build an edgelist when the following conditions are met:

- The source node and target node are not 0, meaning the transition is from one port to another.
- The time spent at a port is greater than a minimum threshold. In this example, 2 hours was used. This limits false positives when a ship transits near a port.

This edgelist can now be used to generate a network. Note that our example from the previous slide is not recorded here since it was a transition from open ocean to Boston. However, the departure from Boston to Newark is recorded.