Structured Data Challenge Report
AI team

Alexis Bogroff*

June 6, 2021

Keywords: Big Data - Computational Complexity - GeoData - Anomaly Detection - Security

1 Big Data, fast queries

The goal is to design a database that would store a large set of GeoData (latitude, longitude, altitude) to track positions of aircrafts. The case specifically requires to optimize range search queries over insertions. In other words, it is desired to build a fast information retrieval system with low memory usage.

Range search is a process selecting the data points that intersect with the range query Q. In less formal terms, it compares points with the lower and upper boundaries defined in Q. The most trivial algorithm would test each stored data point, but the computational (time) cost, referred as *big O notation* $\mathcal{O}(n)$, would be linear in the data size, which is impracticable for building responsive big databases.
The computational complexity $\mathcal{O}(n)$ can represent varying concepts, and in the current case refers to Time and Space complexities. In the following, the focus is on the Time complexity, which can be understood as the number of operations required to obtain a final result (Space complexity represents the memory requirements). In a database, these operations correspond to traversing the nodes of the Tree index, which consist of comparing the encountered index values with the set of conditions in Q. In other words, the complexity corresponds to the expected number of operations as defined informally by: $num\_op = n\_nodes\_traversed * n\_elements\_by\_node\_traversed$. The main objective is to minimize this time, and specifically prevent the time from exploding with an increasing size of data. This is often done by segmenting the data into clusters, and by preventing from traversing clusters that should not contain the requested data.

Algorithms designed for such purposes make use of Tree-based indexing methods, and some are particularly good at indexing geo-spatial data (Sveen (2019)) like the widespread R-Tree method (Guttman (1984)) and its improved and more recent versions (R+-Tree Sellis et al. (1987), Hilbert R-Tree Kamel and Faloutsos (1993), Priority R-Tree Arge et al. (2008), R*-Tree Beckmann and Seeger (2009)). There exist many other common spatial index methods[1]. These indexes might be very time consuming to build (postGIS), but this downside, which is mostly required at insertion time, is negligible since insertions are not frequent in the case.

*mail: alexis.bogroff@gmail.com

[1]Common spatial index methods: Geohash, HHCode, Quadtree, BSP-Tree, etc. - Wikipedia spatial_database

The **complexity of these algorithms** is on average $\mathcal{O}(log_M n)$[2], with M the maximum number of elements by node and n the depth of the Tree index. Each traversed node must be scanned, which explains the M as base for the logarithm. The logarithm intervenes because some nodes leaves are set aside and will not be traversed, which reduce the search space. The greater M the more leaves are set aside, however, M also puts constraints on the memory and seems often set to small values up to five. The scanning process must be repeated n times, along the depth of the Tree. The worst Time complexity however reaches $\mathcal{O}(n)$, which is linear, and might arise if the index Tree can not be optimized for some cases where nodes are overlapping too much and requires the algorithm to scan each node ($\mathcal{O}(log_M(n) * log_M(n)) = \mathcal{O}(n)$).

In the following, a possible **design of the database** will be drawn, after a quick comparison of Structured Query Language systems (SQL) versus Not Only SQL systems. Indeed, as per Bartoszewski et al. (2019), NoSQL (MongoDB) is more efficient (3 times faster) on small to medium size queries (size of gathered data). However, when computing distance from points, the Relational DataBase Management Systems (PostGreSQL) can overcome the former (1000X faster), with MongoDB having issues to scale. Furthermore, very few geo functions exist in the latter, compared to the PostGIS module available to RDBMS. A further study of current solutions could balance in favor of NoSQL systems, which seem to experiment an increasing adoption, and could expect a continuous development of geo functions. The proposed design will correspond to a RDBMS because of a longer personal experience working on these systems.

Main coordinates table structure

Field	Type
date	Timestamp (pkey)
latitude	float
longitude	float
altitude	float

Optional boxes reference table structure

Field	Type
box_id	uuid (pkey)
bound_type	text
latitude	float
longitude	float
altitude	float

- A multi-index must be built upon the tuple (*latitude*, *longitude*, *altitude*) to take profit of the spacial indexing engine and enable the Time complexity discussed beforehand.
- *box_id* can use any other identifier (than uuid) that scales properly and prevents collisions.
- The optional boxes reference table enables the storing of boxes extents.
- *bound_type* corresponds to the type of comparison that should be used. It could be one of *min* or *max*. Boxes should have two entries (one for each bound_type).

[2]Fraser University - Introduction to R-Tree algorithm

2 A long time ago in a galaxy far, far away

2.1 Results, assuming the galaxy was governed by similar laws of physics than Earth!

2.1.1 Areas can be segmented

- Into 2-3 groups (upper-left, lower-right, middle-right) (cf. figure 7)
- A unique area can be identified for TIE Bombers and Droids, but the latter is not very significant as it is based on two data points only (cf. figure 8)
- Two areas can be identified for TIE Fighters at the extremes of the facility (cf. figure 9).
- Imperial Space Shuttles seem spread along side the runway.
- Civilian Star Shuttles are positioned at two spots: one far from other starships on the upper-right, and the second within the TIE Bombers cluster (cf. figure 9).

2.1.2 Many vehicles can be identified as static over multiple periods (cf. figure 15)

- Starships seem static 65% to 80% on average (cf. figure 12).These results highly depend on the floating precision selected, see details in 2.2.2.
- Attack starships (TIE Bombers, TIE Fighters) are more static than Civilian and Imperial Space Shuttles on average (cf. figure 12) and to higher maximal durations (cf. figure 2). Droids have too few points to study such metrics.
- The distribution of static duration has a log-normal like shape, with most starships remaining static less than 5 periods, and very few until +15-40 periods. The right queue of the distribution has a similar shape (cf. figure 18)
- Considering an average of 2 weeks between 2 snapshots ≈50 starships would have been decommissioned on the whole period, i.e. 10 continuously. This seems unlikely on the 3400 detected on the 3 years period. This average gap is indeed poorly representative (see details in section 2.1.4). A static Civilian Star Shuttle apart from the population (top-right) is more likely to be decommissioned.

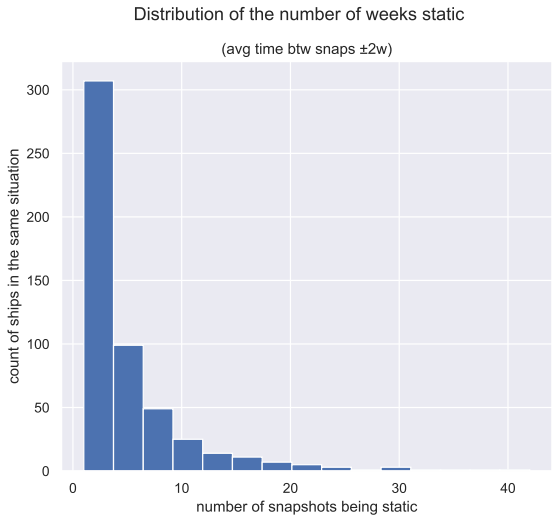


Figure 1

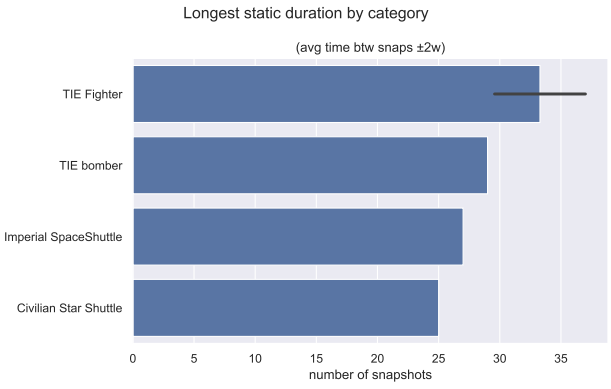


Figure 2

2.1.3 Events and trends

- There is no trend in the count of ships, it is stationary between boundaries (cf. figure 5).
- There is a brutal event around February 2020 with a 75% increase of TIE Fighters count, which might correspond to a purchase (cf. figure 10).

2.1.4 Additional

- Distribution of starships type: +50% TIE Fighters, +20% Imperial Space Shuttles, +10% TIE Bombers, 3% Civilian Star Shuttles (cf. figure 6). One droid only appeared twice.
- The distribution of snapshots over time is not uniform. The number of snapshots range within 0 and 5. Thus, although the average time between two snapshots (considered as *a period* in this report) is not representative of each sub-period. This might also lead to false conclusions concerning the average duration of starships stillness. Indeed, a starship static for 5 periods could represent a static duration ranging within 1 month and 5 months (e.g. 5 snaps are present in both [2018-07-01 - 2028-12-15] and [2018-05] alone). Considering this varying factor of x5, a 12-snapshots-period could range within $\approx$ 2 and 10 months.

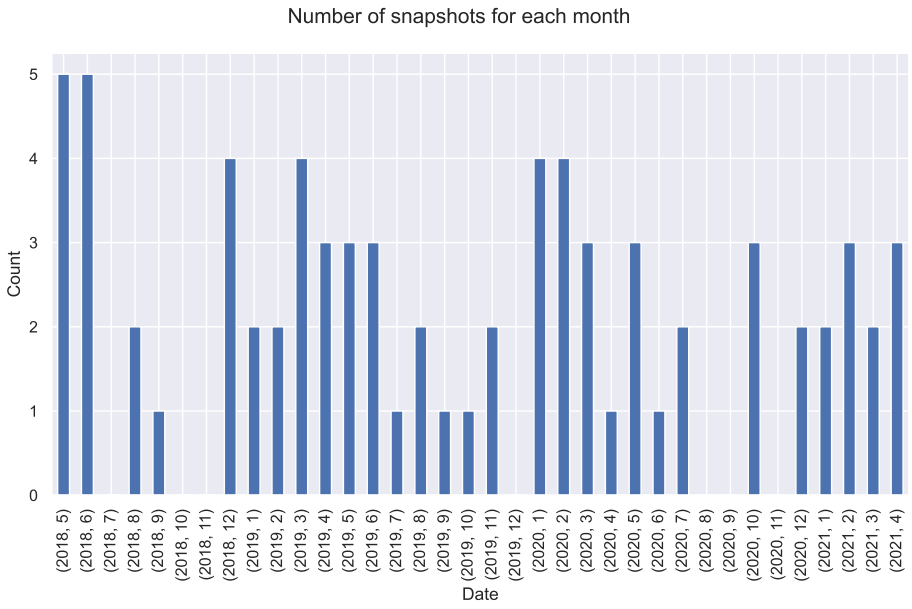


Figure 3

2.2 Methodologies

2.2.1 Area segmentation

Procedures:

- Identify areas of high presence (disregarding starships type):
  1. Compute the distribution of data points location:
  2. Extract the values of the 3 modes from both longitude and latitude
  3. Plot vertical and horizontal lines corresponding the longitude and latitude respectively

4. The intersections are considered as the areas of highest presence (cf. figure 7)

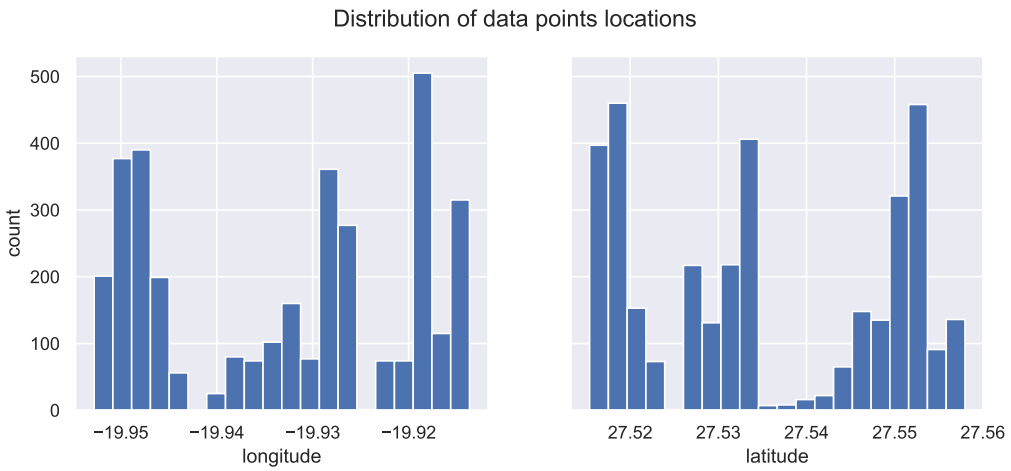


Figure 4

- Identify and locate starships having a unique cluster:
    1. Compute standard deviation for each starship type (on the whole dataset)
    2. Select the type of starships having a standard deviation lower than a (here arbitrary) given threshold, e.g. 0.01 and 0.005 for longitude and latitude respectively (here were selected TIE bomber and Surveillance droid).
    3. Compute the mean longitudes and latitudes of each selected starship type, and consider it a centroid for the category (cf. figure 8)
- Locate clusters for remaining starships:
    1. Assume a number of two centroid per remaining category
    2. Use a K-means clustering algorithm to find appropriate centers (minimize intra-cluster variation, maximize inter-cluster variation) (cf. figure 9). See Section 2.3 for discussion on method limits and possible improvements.

2.2.2 Static detection

Hypothesis:

- The probability of a starship coming back to the exact same location is very low.

Procedure:

1. Select the floating precision: between 1-6 (more info below).
2. Count the number of occurrence of each (exact) value (longitude, latitude)
3. Each (lon, lat) duplicate is considered a static vehicle. The number of occurrences represents the duration of the immobilization.

Additional information

The probability of a starship coming back to the exact same location is actually very dependant on the floating precision of the data point. The higher the floating precision, the lower the probability of encountering the same position.

To visualize this impact, one can compare the results obtained using a precision of 6 (figure 11) vs. a precision of 5 (figure 12) vs. a precision of 4 (figure 13).

This leads to considering the floating precision as a threshold that should be defined conscientiously. It permits to balance between the probabilities of wrongly classifying a starship as static too often vs. too rarely. More precisely, a greater floating precision leads to noise increase, thus:

- Increases the risk of wrongly classifying a starship as moving, while it is actually static.
- Lowers the risk of confounding multiple starships as being the same.

This problematic depends on both:

- the precision of the satellite snapshots (should not be an issue)
- the precision of the computer vision detection algorithm. The lack of robustness of Deep Learning models over slight changes in the input data is a well-known issue, with an active research community developing solutions. In the present case, the variations of the input data could come from variations in:
  - Exposure / light (a Tesla actually hit a truck for this reason)
  - Points blur because of bad weather conditions (clouds)

Which precision to choose?

- The data provided extend from:
  - longitude -20.0 to -19.8627102
  - latitude 27.4990877 to 27.5701161
- Most points range within:
  - longitude -19.95 to -19.91
  - latitude 27.51 to 27.56

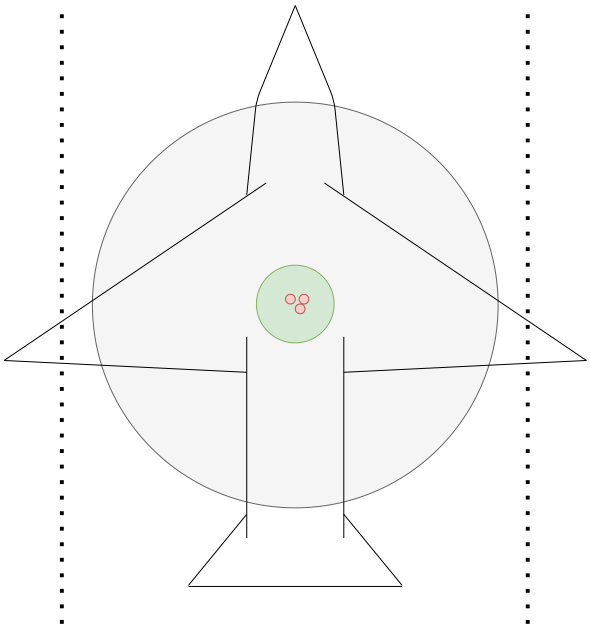
It can be useful to consider the equivalent distance in kilometers:

precision	distance	(lat0, lon0)	(lat1, lon1)
1	14.8647 km	(27.5, -19.9)	(27.6, -20)
2	1.4867 km	(27.5, -19.9)	(27.51, -19.91)
3	148.7 m	(27.5, -19.9)	(27.501, -19.901)
4	14.9 m	(27.5, -19.9)	(27.5001, -19.9001)
5	1.5 m	(27.5, -19.9)	(27.50001, -19.90001)
6	0.1 m	(27.5, -19.9)	(27.500001, -19.900001)

Applied to this case, the points are thus spread along $\approx$ 5kms.

If one considers in the below drawing: a small starship being 10m long; the dotted lines representing its parking lot; the green and red points representing the starship center at different precisions; the following rules could apply:

- It could be assumed that the model would be right to detect the center of the starship in the green zone.
- In case of too high precision (6 floating digits) the center of the starship could eventually be detected as one of the red points. Even though these detections would be equally right, one would like to round-out this noise to rather consider a single center.
- Thus, the imprecision of a computer vision model would be rounded-out if its predictions felt into the green zone (imprecisions that could come from slight variations of light for example).
- Thus, the starship would be considered static if a series of detection would fall in the green zone.
- However, if the starship comes back from a mission, it might well park in a slightly different location. Hence, the center (green zone) would be detected in any place inside the grey zone. The starship would then not be considered static.



2.2.3 Event detection

The explanatory data analysis exploits the entire dataset for its computations, while the programmed detector fits an initial dataset, and then identifies the outliers on a second dataset.

Procedure:

1. Fit:
   (a) Count the number of starships by category over time (on each snapshot) on the initial dataset
   (b) Compute the mean and standard deviation of the starships count by category (what is their central value, how their counts fluctuate around it over time)
   (c) Define lower and upper boundaries: [mean - threshold * std; mean + threshold * std]. The threshold could be around 2-3 (set arbitrarily, but a more robust implementation could use confidence intervals of the empirical law).
2. Detect:
   (a) Count the number of starships by category over time (on each snapshot) on the new dataset

(b) Compare these counts with the boundaries of the corresponding category. The category is considered abnormal on a snapshot if its count is greater than the upper-boundary or lower than the lower-boundary.

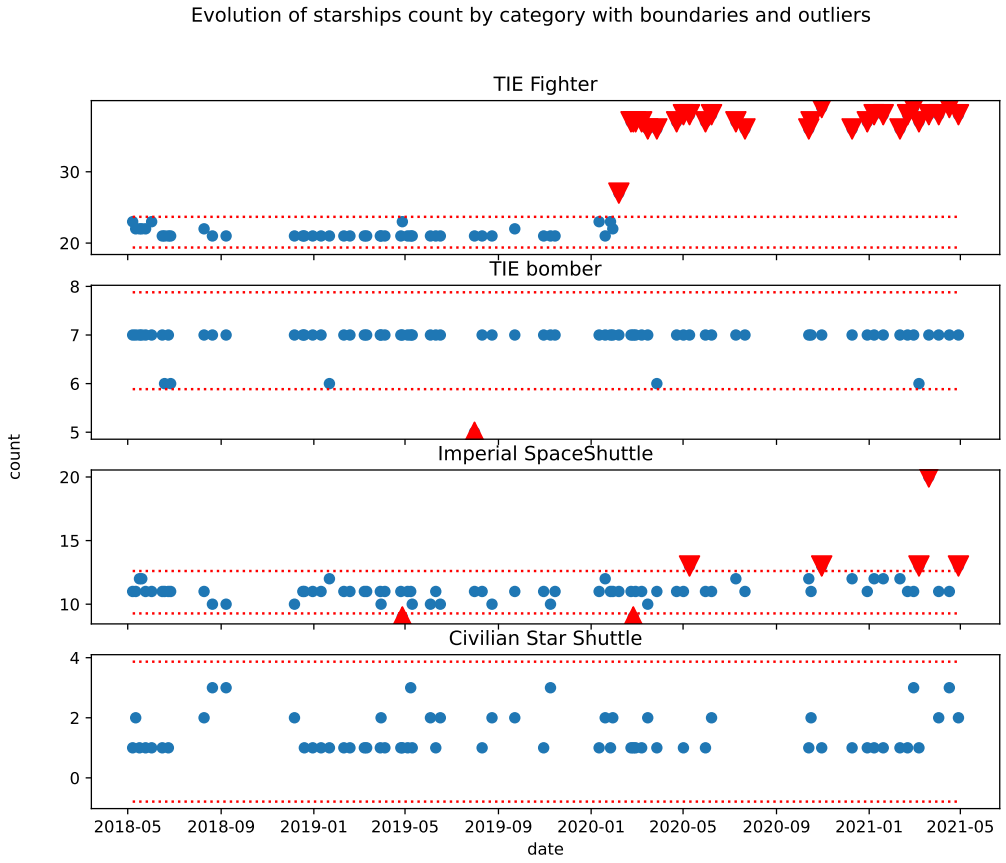


Figure 5

2.3 Discussion

2.3.1 Area segmentation by starship type

The current methodologies lack an automatic selection of the number of centroids, thus one can observe that:

- The Imperial Space Shuttle type have been attributed two centroids that are not coherent with its elongated distribution.
- The Surveillance droid type have been attributed one centroid, which does not seem wrong, but is poorly statistically relevant considering the number of data points (2) of the category.
- The Civilian Star Shuttle suffers from both of these aforementioned issues.

Possible solutions:

- Define the number of centroids using an automated elbow estimation.
- Assess their relevance using statistical tests, to confirm that the centroid(s) is(are) representative of the population.

2.3.2 Data frequency

A higher and more stable snapshot frequency could greatly help in yielding more robust conclusions. Current snapshots are taken on average every two weeks, and the frequency varies greatly from no snapshot to more than four snapshots a month (cf. figure 14). Only persistent stillness can be safely detected. A higher frequency could enable the detection of patterns-of-life on the base and help in the assessment of high activity periods, a potential predictor of sensible information.

2.3.3 Noise correction

Data could be further cleaned up by detecting wrongly classified input categories. E.g., some different starship categories where found within the TIE Fighters clusters. These are potential anomalies that could be clarified with a deeper study.

2.3.4 Robustness and evolution of clusters

Each cluster could be examined using a measure of dispersion around its historical center, e.g. Sum of Squared Differences (SSD). Each snapshot activity could thus be assessed. It would also enable the assessment of the robustness of the centroid, by studying the variation of the SSD over time, or even detect a shift of the central point, i.e. a change in behaviors (distribution shift). Alerts could be easily raised based on an evaluation of the latest activity level compared to an empirical range of normal levels.

References

Arge, L., Berg, M. D., Haverkort, H., and Yi, K. (2008). The priority r-tree: A practically efficient and worst-case optimal r-tree. *ACM Transactions on Algorithms (TALG)*, 4(1):1–30.

Bartoszewski, D., Piorkowski, A., and Lupa, M. (2019). The comparison of processing efficiency of spatial data for postgis and mongodb databases. In *International Conference: Beyond Databases, Architectures and Structures*, pages 291–302. Springer.

Beckmann, N. and Seeger, B. (2009). A revised r*-tree in comparison with related index structures. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 799–812.

Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57.

Kamel, I. and Faloutsos, C. (1993). Hilbert r-tree: An improved r-tree using fractals. Technical report.

Sellis, T., Roussopoulos, N., and Faloutsos, C. (1987). The r+-tree: A dynamic index for multi-dimensional objects. Technical report.

Sveen, A. F. (2019). Efficient storage of heterogeneous geospatial data in spatial databases. *Journal of Big Data*, 6(1):1–14.

2.4 Appendices

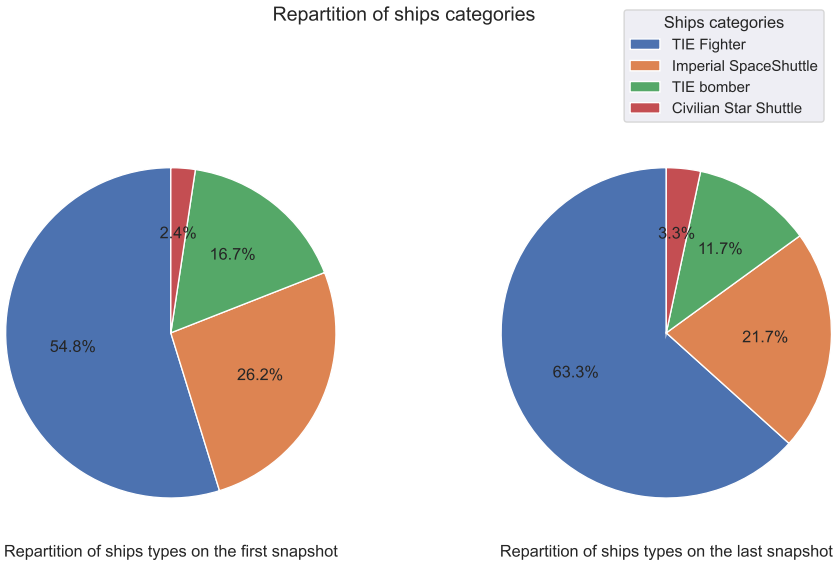


Figure 6

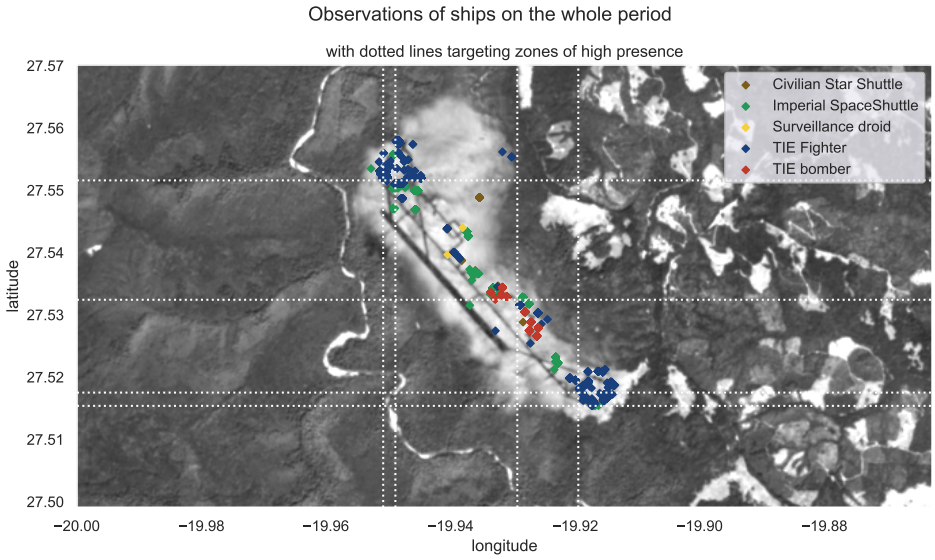


Figure 7

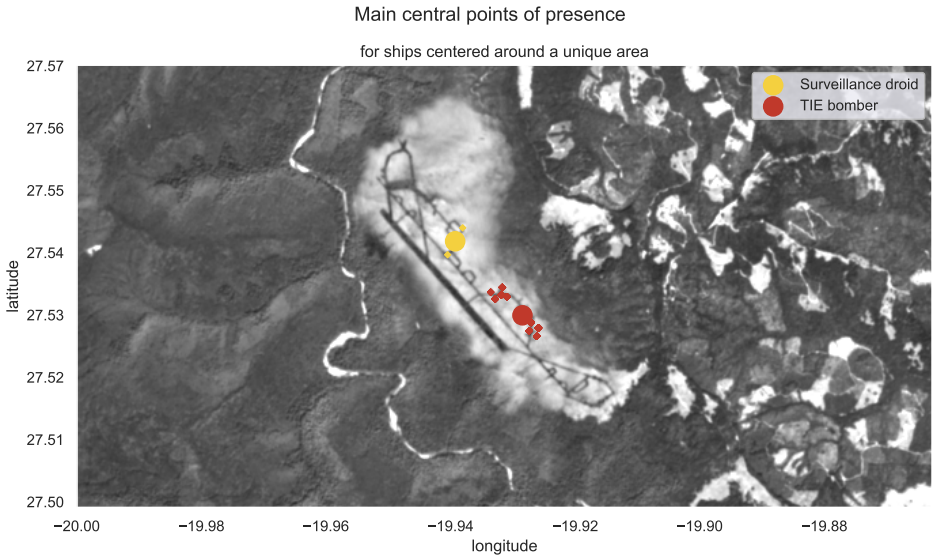


Figure 8

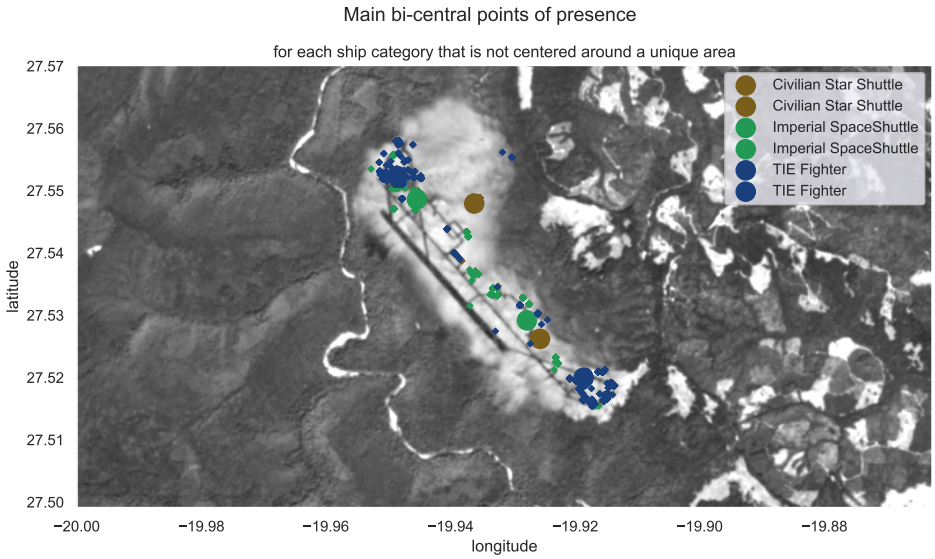


Figure 9

Figure 10

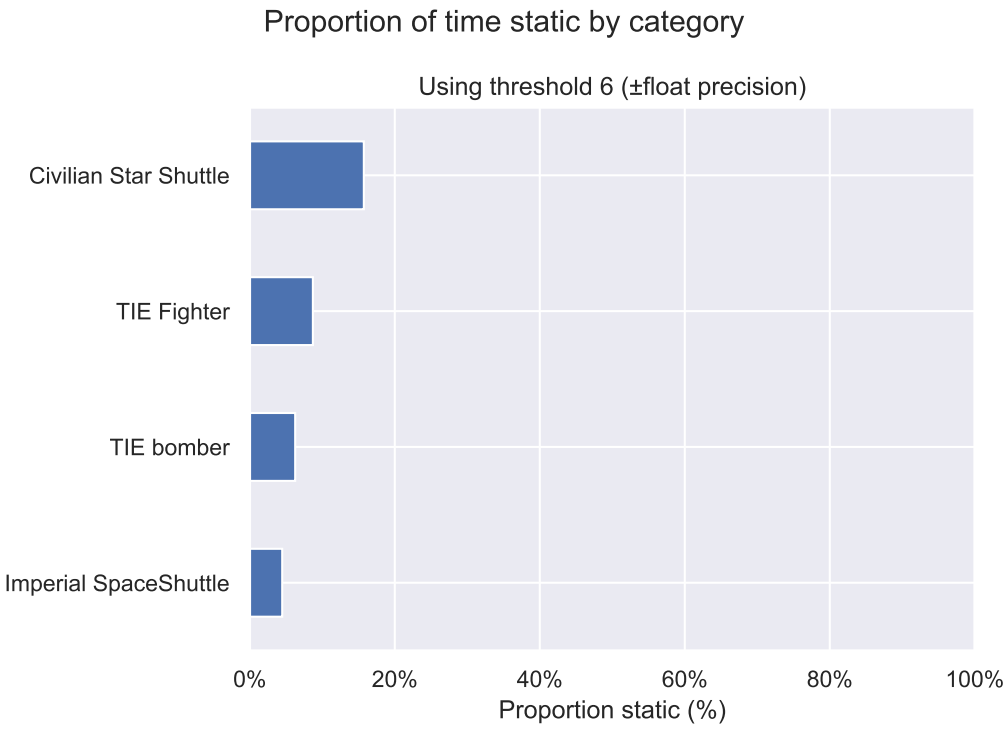


Figure 11

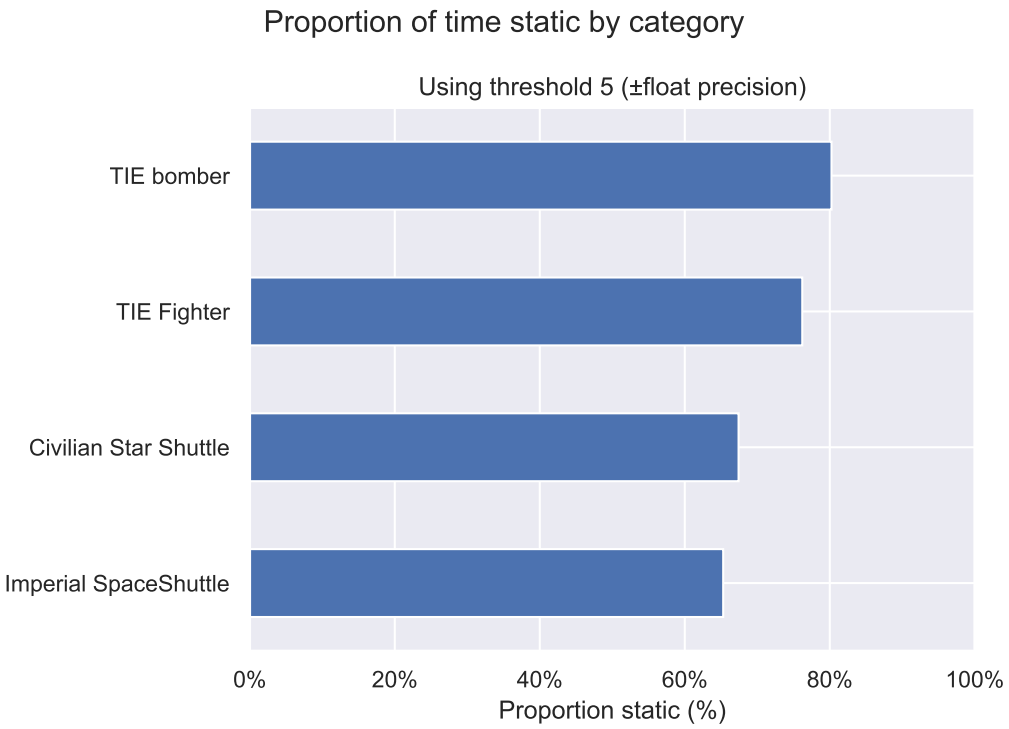


Figure 12

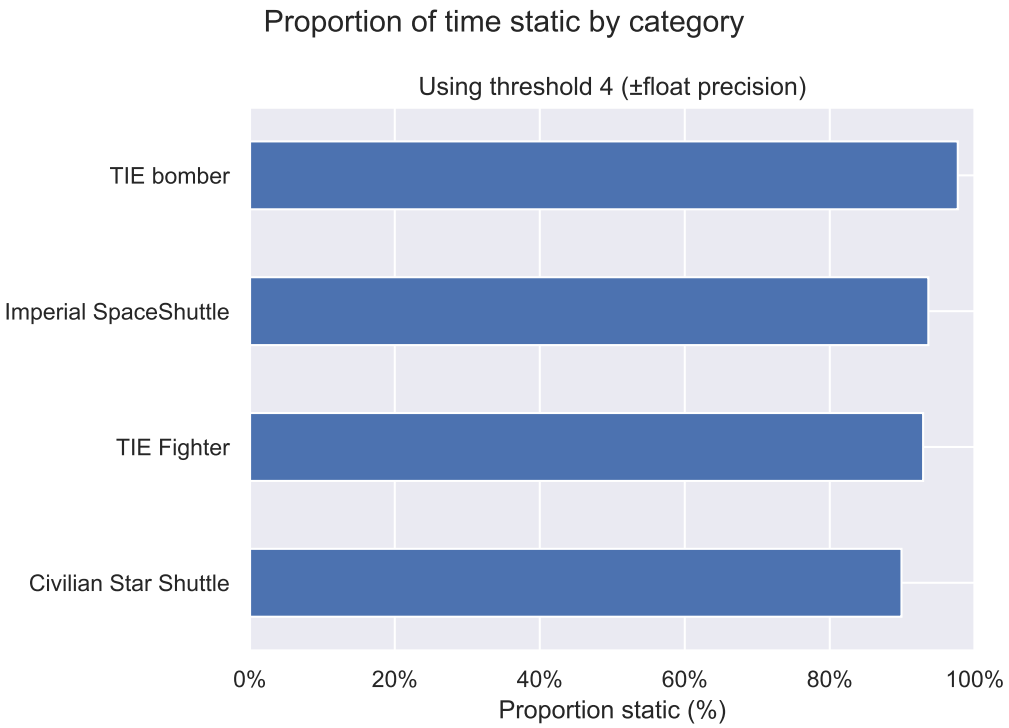


Figure 13

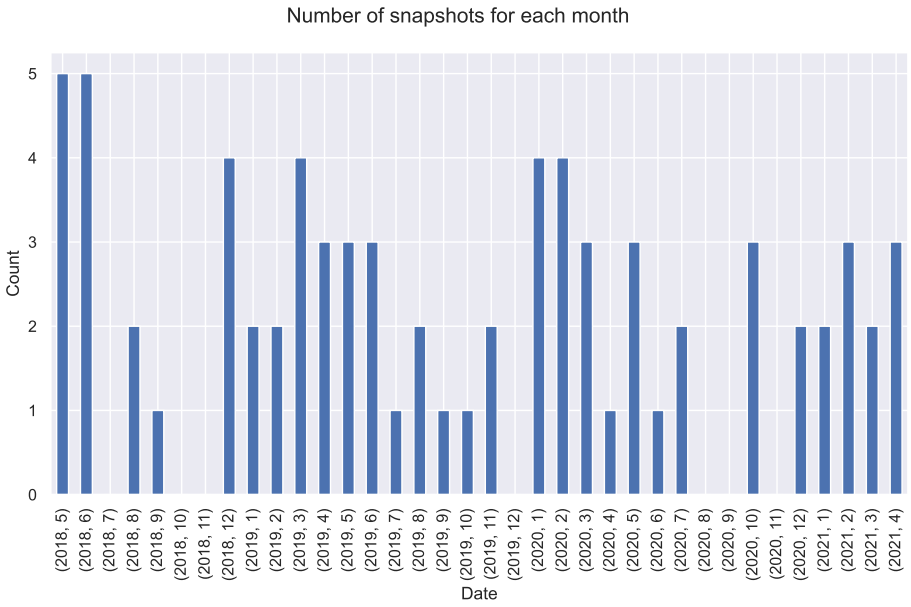


Figure 14

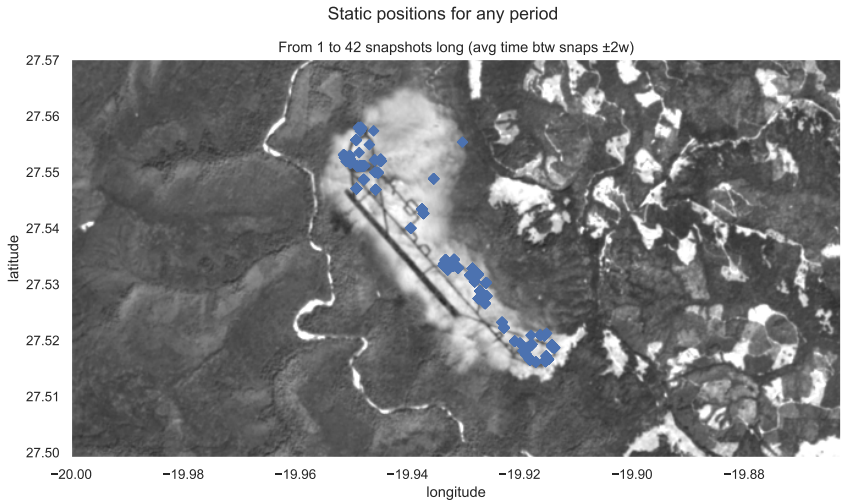


Figure 15

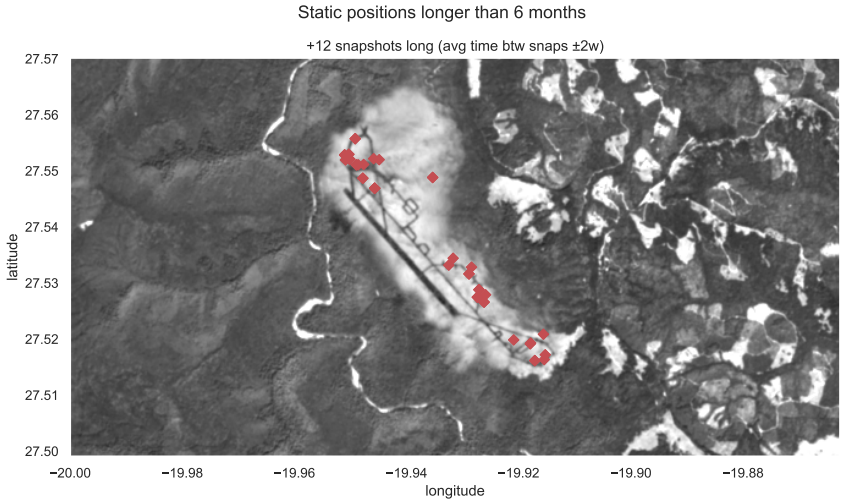


Figure 16

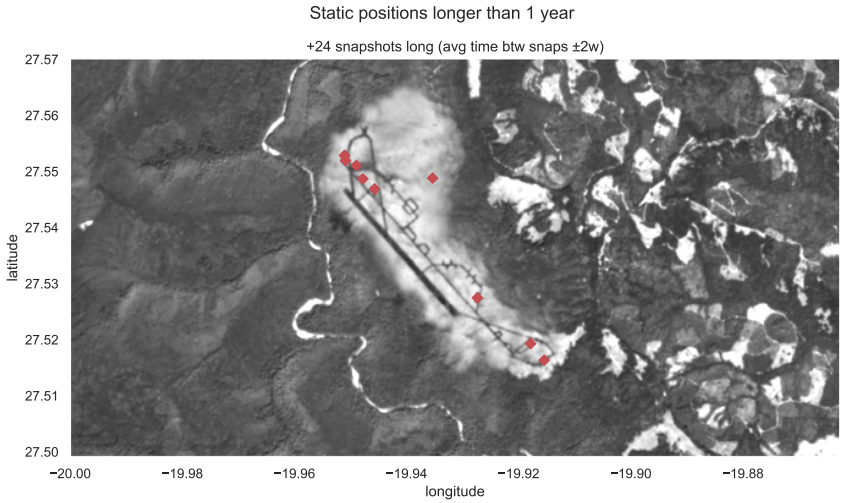


Figure 17

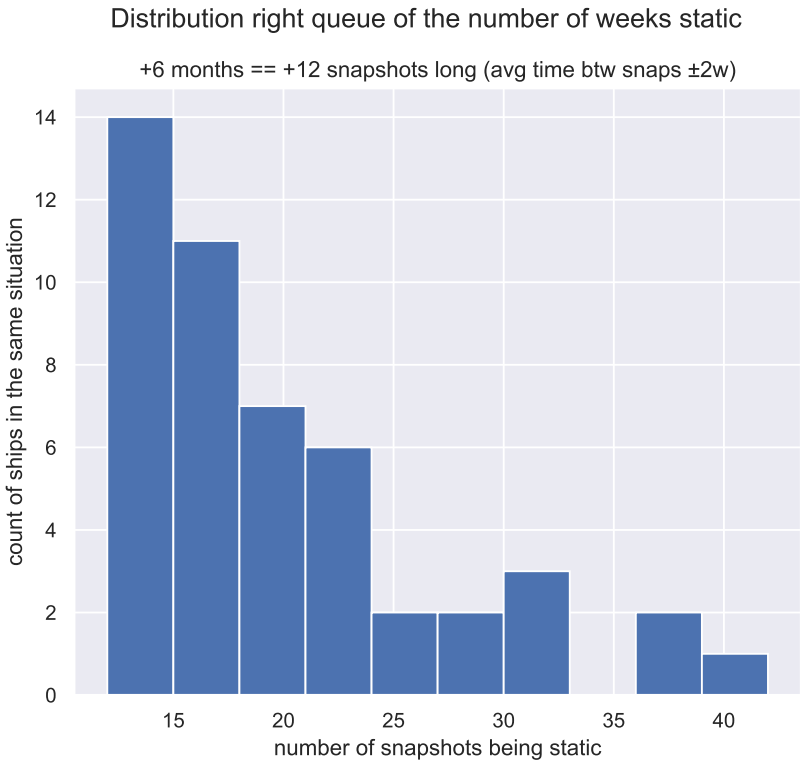

Distribution right queue of the number of weeks static
+6 months == +12 snapshots long (avg time btw snaps ±2w)
count of ships in the same situation
0
2
4
6
8
10
12
14
15
20
25
30
35
40
number of snapshots being static

Figure 18