HEXAGONAL GRIDS APPLIED TO CLUSTERING LOCATIONS IN WEB MAPS

A. Beresnev [1, *], A. Semenov [2], E. Panidi [1]

[1] Department of Cartography and Geoinformatics, Institute of Earth Sciences, Saint Petersburg State University, St. Petersburg, Russia - st040742@student.spbu.ru; artembert@gmail.com
[2] Geosemantica LLC, St. Petersburg, Russia - info@geosemantica.ru

KEY WORDS: Clustering, Hexagonal Grid, Visualization, Geographic Data.

ABSTRACT:

One of the popular ways to clutter reduction techniques is to combine neighboring points into one marker that somehow shows that it contains multiple entities – this way is called clustering. In this paper, we present a JavaScript library to define optimal size of clusters and render them. Moreover, markers have to present heterogeneous data inside of clusters.
The presented library relies on server side clustering, no matter if is it a real-time clustering or a static bunch of hexagonal grids. For the library, a server provides the bunch of grid layers by different cell sizes – from smaller to larger. The library relies on data fetching provided by external library, such as Mapbox/Maplibre, so it can work with both GeoJSON and vector tiles. Using the HTML Canvas to render the marker allows to full customizing the marker image: manage the colors and proportions of cluster fractions and the size.

1. INTRODUCTION

Displaying markers on the Web map is one of the easiest tasks that require only latitude and longitude of giving point. However, by multiplying the number of points in the small area, close markers are going to overlap each other, and the map losing its readability. One of the popular ways to clutter reduction techniques is to combine neighbouring points into one marker that somehow shows that it contains multiple entities – this way is called clustering. Most popular Web maps libraries (*Google Maps*, *Mapbox*, *ESRI*) implement clustering right in the browser, which causes severe performance degradation, especially on large amounts of data. We propose to combine points into clusters on server-side to increase the performance and responsibility of Web map. On the server-side, we used a same-size hexagonal grid that is often used to analyse various spatial data.

1.1 Terms and concepts

Zoom level – a number between 0 and 23 (or 24) that defines how large or small the contents of a map appear in a map view. Unlike paper maps, which have a fixed scale, Web maps are displayed on different displays with different pixel densities. The zoom level determines the number of meters contained in a pixel. Thus, the zoom level is the digital equivalent of a "paper" scale.

2. BACKGROUND

2.1 Client-side and the server-side clustering

Most popular Web maps libraries implements clustering in client side, it means that server sends all the points to the browser, and the browser made a calculation to group close points together. Client-side rendering algorithm that used in *Google Maps* and *Mapbox* calls ‘Supercluster’ were written by Vladimir Agafonkin (Mapbox, 2022) and Dave Leaver (Leaflet, 2022, Agafonkin, 2014). Their approach calls Hierarchical greedy clustering: it uses spatial index to speed up querying and applies cashing to reduce the calculation on zoom level change (Hexmoor, 2015) (Figure 1).

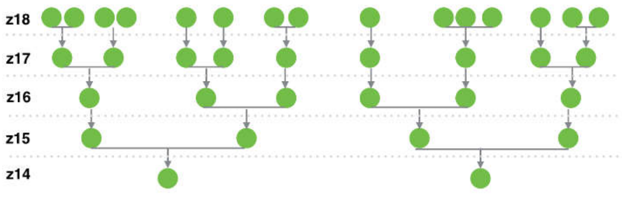

Figure 1. Hierarchical greedy clustering using in Google Maps and Mapbox (Agafonkin, 2016).

Despite significant performance improvements reached by ‘Supercluster’, browser still need to fetch all point from a server. 100,000 points in *GeoJSON* could take at least 10 Mb – browser should spend more than 10 seconds to download it on 3G (HSPA), expecting parsing and calculation.
Moving clustering from browser to server-side provides significant performance improvement for two reasons: 1) network – sending only one point feature for each grid cell instead of all features; 2) CPU – CPU-intensive clustering algorithms do not run in browser anymore, they are running on server side.

2.2 Clustering representation on the map

Popular clustering approaches could be separated into two groups: distance-based approach and anchor-points-based approach. The distance-based approach: hierarchical/agglomerative clustering, k-means clustering (Bharathwaj, 2020). The Distance-based approach relies on optimal combining of points: to merge the nearest points. Anchor-based approach group points around valuable points on a map: cities, countries, sights etc. It is more geographical than distance-based. Voronoi polygons are commonly used to group points around anchors (Figure 2).

* Corresponding author

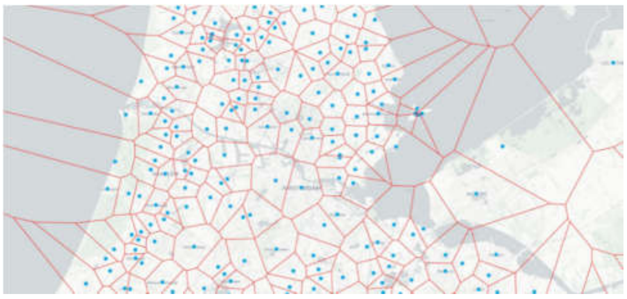

Figure 2. Division of the Netherlands by the Voronoi polygons generated around the cities.

The most significant advantage of the anchor-based approach is that it does not produce false points in the gravity center of the cluster. However, it is required to define anchor points on each zoom level: city centroids could be used as anchor points on country scale, but they do not fit on city scale (Figure 3).

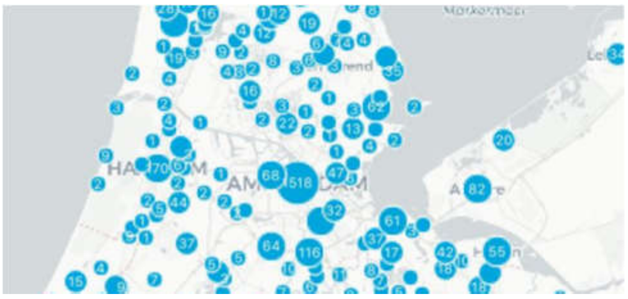

Figure 3. Readability losing on medium zoom level when applying anchor-points-based clustering approach.

2.3 Hexagonal grid

The compromise method to define anchor points for clusters is using a regular grid. In mathematics, a tiling means decomposition of the space into a collection of geometric shapes representing some logical subspaces covering the whole space. There are only three types of regular tiling: Triangular, square and hexagonal. Advantages of hexagon over square and triangle contains:

1. Hexagon has uniform distance between tiles centers
2. Hexagons shape closest to the circle

Both These advantages are significant to render the cluster marker inside the grid cell (Figure 4).

In spite of the triangle and the square, the hexagon is not infinitely decomposable, but this issue could be solved by principles of Gosper fractal that gives hexagons a hierarchical organization. So, it becomes available to use spatial indexing to optimize clustering in different grid sizes (Figure 5).

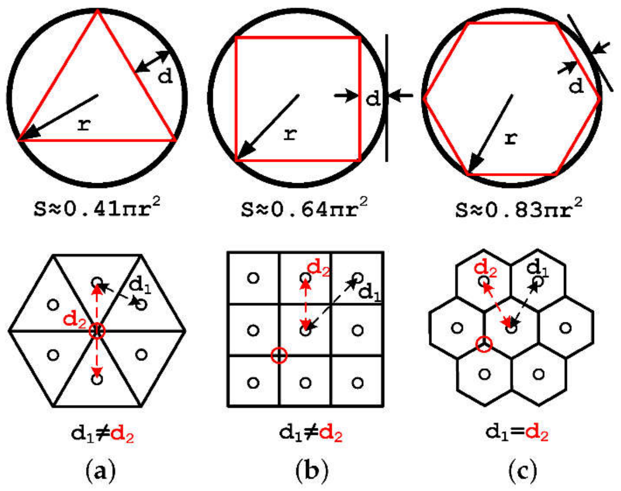

Figure 4. A comparison of properties of regular grids: (a) Triangular, (b) orthogonal, (c) hexagonal (Uher et al., 2019).

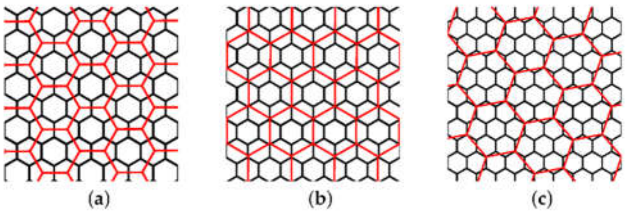

Figure 5. Multi-resolution hexagonal grids: (a) aperture 3, (b) aperture 4, (c) aperture 7 (Uher et al., 2019).

2.4 Existing hexagonal grid frameworks

H3 by *Uber* (Uber, 2022) and *dggridR* (Barnes, 2018) are the most remarkable from the various open-source implementations of hexagon frameworks.
The *H3* grid by *Uber* is based on applying a regular hexagon grid to each face of an icosahedron, and then projecting those faces to the spherical surface of the Earth using inverse gnomonic projection (Figure 6). An icosahedron-based map projection results in twenty separate two-dimensional planes rather than a single plane. The icosahedron can be unfolded in many ways, producing a two-dimensional map each time. *H3*, however, does not unfold the icosahedron to build its grid system, and instead lays its grid out on the icosahedron faces themselves, forming a geodesic discrete global grid system (Brodsky, 2019).

H3 library is developed on *C* language. It provides bindings to multiple languages to make its usage easier. The most popular bindings are *h3-js*[1] – bindings for JavaScript and *h3-py*[2] – for Python.
DggridR by Richard Barnes provides hexagonal, triangular and diamond grids on several projections , but the recommended one by the author is Icosahedral Snyder Equal Area Aperture 3 Hexagonal Grid. This grid, along with the other icosahedral grids, ensures that all cells are of equal area, with a notable exception. At every resolution, the Icosahedral grids contain 12 pentagonal cells which each have an area exactly 5/6 that of the hexagonal cells, that makes it close to *H3* grid by *Uber*.
DggridR library is written in *C++* language and supplies as an *R* language library without additional bindings to another languages, that make its usage harder than *h3* by *Uber*.

[1] *https://github.com/uber/h3-js*

[2] *https://github.com/uber/h3-py*

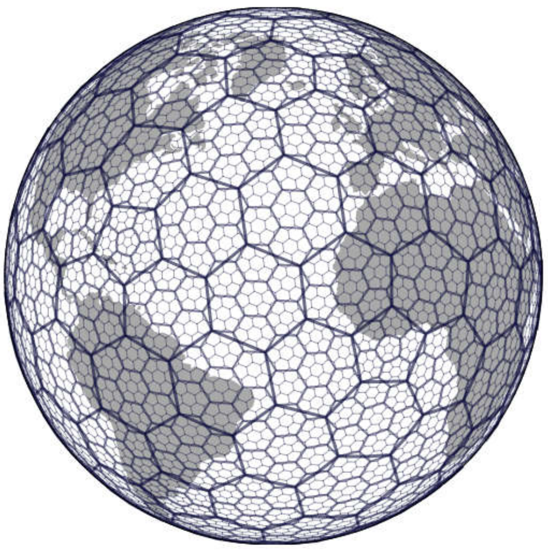

Figure 6. Representation of *H3* grid on a planet scale (Brodsky, 2019).

3. DEVELOPED LIBRARY

In this paper, we present a JavaScript library to define optimal size of clusters and render them.
The goal: present heterogeneous data inside of cluster's marker.

3.1 Requirements

Basing on prospected applying, we define the requirements for developing library:

1. Marker should show different types of data it contains
2. Library has to support integration with *Mapbox/Maplibre* (the most popular JavaScript libraries to show map)
3. Markers have to react on maps events: zooming and dragging – them have to follow the map smoothly
4. Grid sizes are various, library have to adopt for any of them
5. Minimum and maximum zoom level for each grid size should be counted by the library
6. Markers have to show the amount of containing points
7. Markers containing the same number of points have to be the same size during zooming the same grid layer

3.2 Input data

The presented library relies on server side clustering, no matter if is it a real-time clustering or a static bunch of hexagonal grids. For the library, a server provides the bunch of grid layers by different cell sizes – from smaller to larger. It could be a list of given sizes: [2000, 4000, 6000, 8500, 13000…]. The other input value is a minimum size of marker in pixels.

3.3 Expected behavior

A server provides several layers containing grids by different cell size. When the user zooms the map, layers automatically have to change. While user see the same layer, markers have to stay in place in the center of the grid cell. When the user zooms in enough, the map have to hide the current grid layer and show the next, which contains smaller cells – that time markers have to place centers of new cells (Figure 7).

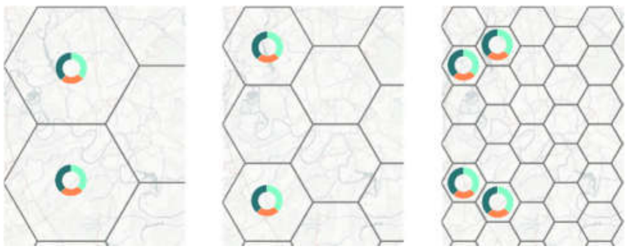

Figure 7. Expected behaviour – sketch.

3.4 Defining zoom levels for given grid layers

The main goal of the library is to define minimum and maximum zoom level for each layer. The min zoom level for each layer is based on visibility and clarity of complex multi-color marker. It's mean that marker should not be smaller than defined number of pixels (that should be defined empirically based on map content and color). If we use a hexagonal grid, the size of the marker is equal to the inscribed circle radius of the hexagon cell (Figure 8).

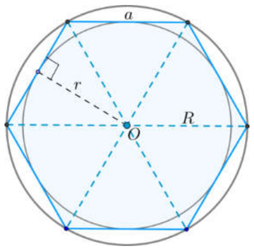

Figure 8. Main dimensions of hexagonal, where r = the inscribed circle radius.

Based on known grid-cell size of levels in meters, the horizontal distance represented by one pixel could be resolved using the formula (1):

$$S_{pixels} = C \times cos(latitude) \div 2^{zoom\ level\ +\ 8} \qquad (1)$$

where, S = horizontal distance in one pixel, m
C = equatorial circumference of the Earth

The maximum zoom level of each layer is the minimum of the bigger one (Table 1).

Grid cell radius, m	min zoom level	max zoom level
2000	11.614	14.000
4000	10.614	11.614
6000	10.029	10.614
8500	9.527	10.029
13000	8.914	9.527
19000	8.367	8.914
32000	7.614	8.367

Table 1. Zoom levels for each layer of hexagonal grid calculated by developed library.

3.5 Rendering of markers

Based on the requirement to embed markers to *Mapbox/Maplibre* library (Porter et al., 2021), and on demand to show different types of data inside of marker, we decided to render markers as

an *HTML5* Canvas elements (Li et al., 2018). Canvas is a technology to render graphic in a Web page as a bitmap. It provides the way to draw geometric figures from *JavaScript* code in a real time. Canvas element can be used as a marker in *Mapbox/Maplibre*.
Proposed *CanvasMarker* is a *JavaScript* function by given interface:

```
function CanvasMarker(
  data: PieChartItem[],
  size: number
): HTMLCanvasElement

interface PieChartItem {
  label: string;
  value: number;
  color: string;
}
```

To prepare the marker, we create the new *HTMLCanvasElement* by given size. Then we render a sector for each type of data containing in this cell, the angular dimension of a given type is proportional to its fraction. When all sectors are ready, we need to clip the center form the pie chart:

```
export function CanvasMarker(
  data: PieChartItem[],
  size = pieChartSizes.diameter
): HTMLCanvasElement {
  const canvas = document.createElement("canvas");
  canvas.width = size;
  canvas.height = size;
  const ctx = canvas.getContext("2d") as
CanvasRenderingContext2D;
  const x = canvas.width / 2;
  const y = canvas.height / 2;
  const total = getTotal(data);

  let startAngle: Radian;
  let endAngle: Radian;

  for (let i = 0; i < data.length; i++) {
    startAngle = calculateStartAngle(data, i, total);
    endAngle = calculateEndAngle(data, i, total);

    ctx.beginPath();
    ctx.fillStyle = data[i].color;
    ctx.moveTo(x, y);
    ctx.arc(x, y, y, startAngle, endAngle);
    ctx.fill();
  }
  clipCenterCircle(ctx);

  return canvas;
}
```

3.6 Intergation with Mapbox/Maplibre

Mapbox/Maplibre libraries provide a rich API to embed custom markers to given source layers[3]. The developed library can be easily implemented into the map:

```
const addMarkers: (
  map: maplibregl.Map,
  clustersLayersNames: string[],
  layer: ClustersSourceMetadata
) => void = (map, clustersLayersNames, layer) => {

  const visibleFeatures = map.querySourceFeatures(layer.table,
{
    sourceLayer: layer.id,
  }) as any as ClusterFeature[];

  const visiblePoints = filterClusterPoints(visibleFeatures);
  const [minCount, maxCount] = [
    Math.min(...visiblePoints.map((item)                    =>
parseInt(item.properties.num, 10))),
    Math.max(...visiblePoints.map((item)                    =>
parseInt(item.properties.num, 10))),
  ];

  visiblePoints.forEach(({ geometry, properties }) => {
    const marker = new maplibregl.Marker({
      element: CanvasMarker(
        preparePieChartData(properties),
        resolveClusterSizeInLayerRange(
          parseInt(properties.num, 10),
          minCount,
          maxCount
        )
      ),
    });
    marker.setLngLat(geometry.coordinates).addTo(map);
    existingMarkers[getUniqueId(layer.id,      properties)]     =
marker;
  });
};

export function filterClusterPoints(
  points: ClusterFeature[]
): ClusterFeature[] {
  return points.filter((item) => {
    if (item.geometry.type !== "Point") {
      return false;
    }
    if (!item.properties.id || !item.properties.num) {
      return false;
    }
    return true;
  });
}
```

4. RESULTS AND DISCUSSION

The developed library provides the way to render markers for clusters over the hexagonal grid. The library relies on data fetching provided by external library, such as *Mapbox/Maplibre*, so it can work with both *GeoJSON* and vector tiles. Using the *HTML Canvas* to render the marker allows to full customizing the marker image: manage the colors and proportions of cluster fractions and the size. Comparing to existing solutions, like *Recyclemap.ru* by *Greenpease*, the presented library show the content of clusters and visually distinguish clusters by the amount of contained points (Figure 9).

[3] *https://maplibre.org/maplibre-gl-js-docs/api/markers/#marker*

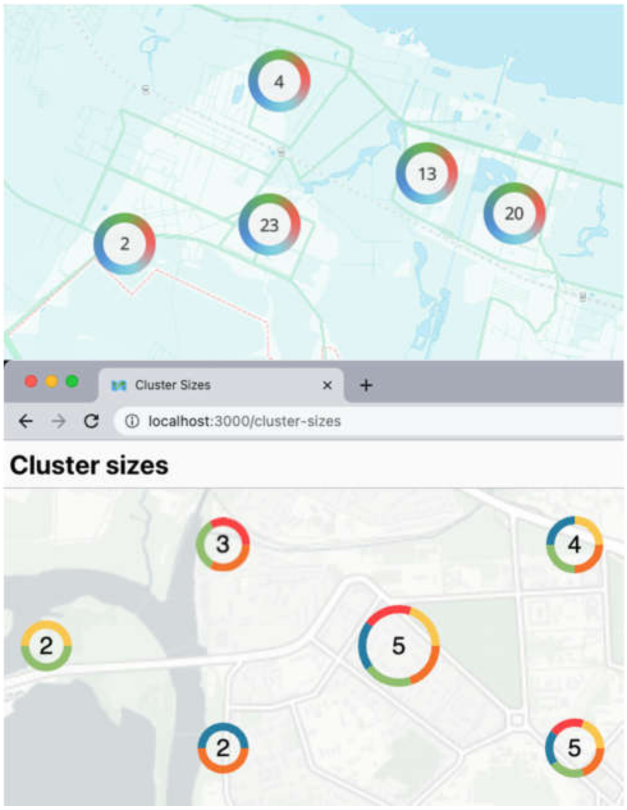

Figure 9. Comparison of pie charts visualization on *Recyclemap.org* (top) and map generated by developed library (bottom).

4.1 Limitations

Transformation from map projection coordinate system (meter units) to screen coordinate system (pixel units) leads to coordinate distortions, which grows inversely with the scale, so we cannot recommend to use hexagonal grids on a small scale (Figure 10).

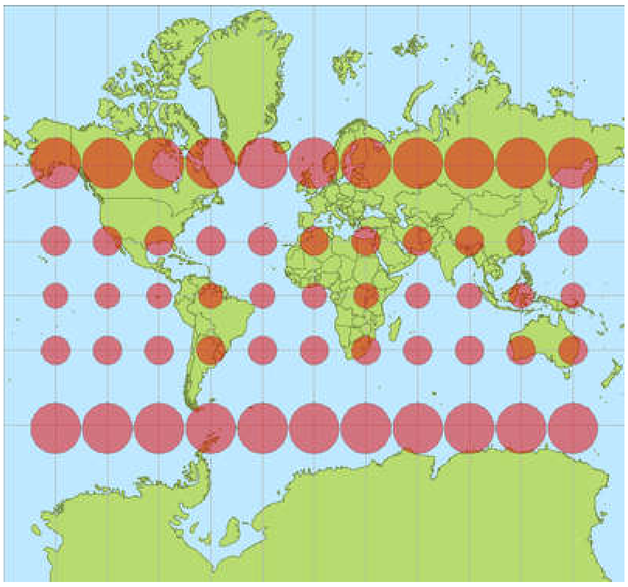

Figure 10. Variation in metres per pixel with latitude on the Mercator projection. Sizes of circles are incorrect; they should be opposite (OpenStreetMap Wiki, 2022).

Proposed solution contains two steps: clustering on the server-side (1) and the use of hexagonal grid approach (2) instead of distance-based or anchor points-based algorithms. Any configurations that were available on the client side now should be implemented on the server side: such as grid cell size changing or adding/removing points features to data source.
Focusing on the hexagonal grid in comparison to other clustering algorithms, the regular grid cells have strictly determined sizes, so we should prepare the series of grids for each zoom level.

REFERENCES

Agafonkin, V., 2014: Leaflet: an open-source JavaScript library for mobile-friendly interactive maps. https://leafletjs.com Accessed January 9, 2022

Agafonkin V., 2016: Clustering millions of points on a map with Supercluster. Mapbox blog: https://blog.mapbox.com/clustering-millions-of-points-on-a-map-with-supercluster-272046ec5c97 Accessed January 9, 2022

Bharathwaj, M., 2020: Towards data science: Clustering Techniques. Available online: https://towardsdatascience.com/clustering-techniques-hierarchical-and-non-hierarchical-b520b5d6a022 Accessed January 9, 2022

Birch, C.P.D., Oom, S.P., Beecham, J.A., 2007: Rectangular and hexagonal grids used for observation, experiment and simulation in ecology. *Ecological Modelling*, 206(3), 347-359. doi.org/10.1016/j.ecolmodel.2007.03.041

Barnes, R., 2018: R-Barnes/Dggridr: V2.0.3. Zenodo. doi.org/10.5281/ZENODO.1322866

Brodsky, I., 2019: H3: Uber's Hexagonal Hierarchical Spatial Index. Available online: https://eng.uber.com/h3/ Accessed January 9, 2022

Burdziej, J., 2011: A Web-based spatial decision support system for accessibility analysis—concepts and methods. Applied *Geomatics*, 4(2), 75-84. doi.org/10.1007/s12518-011-0057-x

Hexmoor, H., 2015: Diffusion and Contagion. In: Computational Network Science. Elsevier. 45-64. doi.org/10.1016/b978-0-12-800891-1.00006-8

Mapbox, 2022: supercluster. GitHub repository: https://github.com/mapbox/supercluster Accessed January 9, 2022

Leaflet, 2022: Leaflet/Leaflet.markercluster. GitHub repository: https://github.com/Leaflet/Leaflet.markercluster Accessed 9, 2022

Li, D., Mei, H., Shen, Y., Su, S., Zhang, W., Wang, J., Zu, M., Chen, W., 2018: ECharts: A declarative framework for rapid construction of web-based visualization. *Visual Informatics*, 2(2), 136-146. doi.org/10.1016/j.visinf.2018.04.011

Lu, C.-A., Chen, C.-H., Cheng, P.-J., 2011: Clustering and Visualizing Geographic Data Using Geo-tree. Presented at the 2011 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT). doi.org/10.1109/wi-iat.2011.171

OpenStreetMap Wiki. Zoom levels. https://wiki.openstreetmap.org/wiki/Zoom_levels Accessed January 9, 2022

Porter, M. E., Hill, M. C., Harris, T., Brookfield, A., Li, X., 2021: The DiscoverFramework freeware toolkit for multivariate spatio-temporal environmental data visualization and evaluation. *Environmental Modelling & Software*, 143, 105104. doi.org/10.1016/j.envsoft.2021.105104

Rezaei, M., Franti, P., 2018: Real-Time Clustering of Large Geo-Referenced Data for Visualizing on Map. *Advances in Electrical and Computer Engineering*, 18(4), 63-74. doi.org/10.4316/aece.2018.04008

Uber, 2022: h3. GitHub repository: https://github.com/uber/h Accessed January 9, 2022)

Uher, V., Gajdoš, P., Snášel, V., Lai, Y.-C., Radecký, M., 2019: Hierarchical Hexagonal Clustering and Indexing. *Symmetry*, 11(6), 731. doi.org/10.3390/sym11060731