Demers Cartogram with Rivers

Qiru Wang[1], Robert S. Laramee[1]

[a]*Visualization and Graphics Group, University of Nottingham, University Park, NG7 2RD, Nottingham, United Kingdom*

Abstract

Cartograms serve as representations of geographical and abstract data, employing a value-by-area mapping technique. As a variant of the Dorling cartogram, the Demers cartogram utilizes squares instead of circles to represent regions. This alternative approach allows for a more intuitive comparison of regions, utilizing screen space more efficiently. However, a drawback of the Dorling cartogram and its variants lies in the potential displacement of regions from their original positions, ultimately compromising legibility, readability, and accuracy. To tackle this limitation, we propose a novel hybrid cartogram layout algorithm that incorporates topological elements, such as rivers, into Demers cartograms. The presence of rivers significantly impacts both the layout and visual appearance of the cartograms. Through a user study conducted on an Electronic Health Records (EHR) dataset, we evaluate the efficacy of the proposed hybrid layout algorithm. The obtained results illustrate that this approach successfully retains key aspects of the original cartogram while enhancing legibility, readability, and overall accuracy.

Keywords: Geospatial visualization, Cartogram, Demers Cartogram

1. Introduction and Motivation

Cartograms are representations of geographical and abstract data based on a value-by-area mapping combining statistical and geographical information [**?** **?**]. Various styles of cartograms have been proposed and implemented for applications such as urban planning [**?** **?**], natural hazard forecasting [**?** **?**], conservation and environmental planning [**?** **?**], political and social demographics [**?** **?**], and decision-making for public health [**?** **?**].

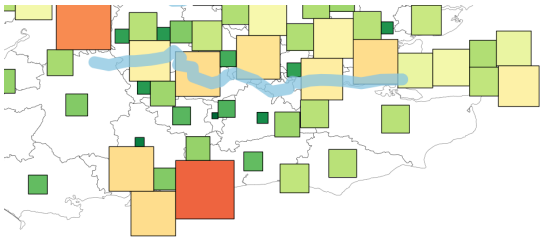


Figure 1: The River Thames passing through a cartographic representation of the NHS CCGs in London and surrounding areas.

Among the four types of cartogram categorized in a survey by **?**] (See **??** for definitions of contiguous, non-contiguous, rectangular, and Dorling), a trade-off is made between types of accuracy (See **??**). For this project, we focus on non-contiguous cartograms like Demers cartograms because they facilitate statistical comparison between regions, they can make good use of screen space, and comparison of regions is useful when studying Electronic Health Records (EHR) data. Demers cartograms offer the advantage in cases where the data is not directly correlated to region sizes. In other words, Demers cartograms are useful when a data dimension is not describing the geography of the region it represents but is tied to something else, for example the health of its population. Also, the comparison of magnitudes becomes an area estimation task, which is effective for a numeric data encoding [**?**]. See **?** [**?**] for a more complete description of the advantages that Demers cartograms offer. Building on Demers cartograms [**?**], we introduce and develop novel features, such as rivers, aiming to improve the readability and geographical accuracy without sacrificing statistical accuracy. Standard Demers cartograms are composed of square nodes that representing geographic enumeration units. As such, this can reduce their legibility. We implement a new hybrid cartographic layout algorithm that combines rivers with the placement of the nodes representing geographic enumeration units, in this case, a Clinical Commissioning Group (CCG). We hypothesize that introducing rivers improves the overall legibility of a cartogram. By *legibility* we mean readability and ability to interpret the cartogram. To assess this hypothesis, we designed an experimental setup where participants engaged in correspondence and location tasks as part of a user study. To reduce error and make efficient use of screen space, the algorithm also updates the position of rivers to accommodate the node layout. We then apply the algorithm to a real-world case study using EHR data to evaluate the result. We present a user study that demonstrates its effectiveness.

Our contributions include:

- A new variant of Demers cartograms that incorporates rivers to improve readability and recognizability,
- A novel hybrid layout algorithm that combines node positions with features such as rivers,
- A user study evaluation of the technique with an application to EHRs.

The results of the user study indicate that rivers can improve the legibility of cartograms. One of the major challenges involved is developing a layout algorithm that handles different shapes. In other words, the hybrid layout algorithm is novel because it handles different types of elements: square node representing regions and polylines representing rivers. Another challenge we overcome in developing the algorithm is to resolve stalemate situations where nodes become congested due to constraints imposed by rivers, while ensuring error minimization.

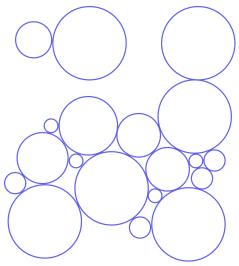
2. Related Work

This section introduces the characteristics of various cartogram styles, describes relevant applications of cartograms, and provides a brief overview of some real-world implementations of cartogram-based visualizations.

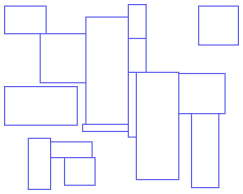
2.1. Definitions

While we focus on rectangular cartogram variants, we start with brief definitions of contiguous and non-contiguous cartograms: Contiguous cartograms preserve topology, maintain connectivity with their adjacent neighbors, but are also subject to distortion in shape. Non-contiguous cartograms sacrifice topological connectivity with neighbors to enable expansion or reduction in size while maintaining their polygonal shape [**?**].

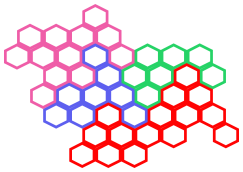
?] define and summarize three major accuracy dimensions for cartograms: statistical, geographical, and topological. Each cartogram design may make various types of accuracy trade-offs between dimensions. We provide a comparison of these trade-offs as introduced by **?**] in **??**. In addition, we include the Demers as it is the focus of our work.



Dorling cartograms, as a variant of non-contiguous cartograms, generally do not preserve geography and topology. A Dorling cartogram is statistically accurate, regions are represented by circles and the data dimensions of interest are represented by the circle area [**?**]. In a Demers cartogram, a variant of Dorling, squares are used instead to capture a certain level of topology, as described by **?**] in their related work section. Dorling cartograms are unable to maintain topological accuracy as circles are often repositioned to remove overlap. Here we focus on Demers cartograms as we use squares to depict regions. This style of cartogram offers the advantages that the comparisons between regions are intuitive and screen space utilization is more efficient. This is important in our use case scenario involving EHRs. Demers cartograms, where regions are represented by squares, often have the advantage of preserving a higher level of topology at the cost of geographical accuracy [**?**].



Rectangular cartograms are contiguous and do not preserve geographical accuracy [**?** **?**]. Depending on the variant, a rectangular cartogram may trade-off between statistical and topological accuracy.



Mosaic cartograms usually use square or hexagonal tiles to depict regions, and are contiguous and sacrifice statistical accuracy to preserve some level of geographical accuracy [**?**]. Some variants are able to preserve topological accuracy as well.

2.2. Peer-reviewed Applications

There has been a substantial amount of research done in this area, here we review some of the important work that has inspired our work.**?**] use a Dorling cartogram to represent religious diversity in the US. **?**] depict 1996 US election data and 2005 Chinese population data using Dorling, Mosaic, and contiguous cartograms. **?**] present a Dorling cartogram to illustrate COVID-19 infections in China. **?**] use a Demers cartogram to visualize health-related data by CCG regions in England, the work introduces a novel technique to remove the overlap of squares based on topological features, aiming to improve both geographical and topological accuracy. **?**] investigate the memorability of contiguous and Dorling cartograms using multiple data sets that include demographic, agriculture, and retail data in the US. See **??** for a list of literature that adopts cartograms for visualization with corresponding geographical regions and node counts.

Our work extends the algorithm described by **?**] which incorporates a static topological feature into Demers cartograms. Our work enhances that of **?**] in multiple ways. First, we introduce multiple features (rivers) into the layout, as opposed to a single river. Second, we make topological features dynamic and further improve legibility and geographical accuracy. By the term *dynamic*, we mean that the position of the rivers is updated as part of the layout algorithm. In previous work, the river is static and serves merely as a boundary. Third, we improve the algorithm to resolve stalemates. Finally, the way we evaluate the cartograms is also different. **?**] count river crossings to evaluate error (a statistical metric). Here our focus is on readability, and thus we include a user study.

2.3. Cartograms in Media

Cartograms are an engaging visual representation and therefore they are a popular choice of representation in covering various topics by the media. The Washington Post uses cartograms to visualize the US overseas economic assistance, in arm sales (Mosaic) [**?**], the 2016 US Election (contiguous) [**?**], and the Brexit Referendum (Mosaic) [**?**]. National Geographic uses contiguous and Mosaic cartograms to analyze the 2016 US Election results [**?**], the same topic is also covered by the Financial Times with a Dorling cartogram [**?**]. **?**] adapts a Dorling cartogram with both contiguous and non-contiguous

Literature	Cartogram Type(s)	Geographic Region(s)	Number of Nodes	Year
?]	Dorling	US	3,142	?
?]	Dorling, Mosaic, Neighbor-preserving	US, China	34 - 49	?
?]	Dorling, Non-contiguous, Neighbor-preserving	Portugal	2,882	?
?]	Demers	England	209	?
?]	Dorling	China	34	?
?]	Contiguous, Dorling	US	49	?
?]	Non-contiguous, Demers	US, Netherlands, World	49 - 342	?

Table 1: Related work with non-contiguous cartogram-based visualizations. **Cartogram type** is the type of cartogram used. **Geographic region** is the geographic region depicted by the cartogram. **Number of nodes** is the number of nodes (representing geographic enumeration units) depicted in the cartogram.

	Accuracy			
Cartogram Variant	**Statistical**	**Geographical**	**Topological**	**Contiguity**
Contiguous	Variable	Variable	Accurate	Yes
Non-contiguous	Accurate	Shape is accurate	Inaccurate	No
Rectangular	Variable	Shape is inaccurate	Variable	Yes
Dorling	Accurate	Inaccurate	Variable	No
Demers	Accurate	Inaccurate	Variable	No

Table 2: Trade-off between dimensions. Dimension sacrificed in order to improve target dimension's accuracy.

cartograms to represent the gender pay gap in Portugal. ?] reports the 2018 US midterm Election with a Mosaic cartogram, the same approach is used to cover the 2020 US Election by the New York Times [?] and Bloomberg [?].

One of the disadvantages of Dorling and Demers cartograms is legibility. The layout algorithms may displace regions far from their original position and make the maps more difficult to interpret. ?] present a method to compute stable Demers cartograms with multiple constraints to maintain adjacencies with no overlapping nodes.

In this paper, we introduce a new type of topological feature, a river, as a constraint to compute the final layout, with the aim of improving the interpretation, readability, and accuracy of this class of cartograms.

3. Data Description

Processing heterogeneous data can be challenging, especially when an EHR dataset is involved, because the data comes from multiple sources [?]. The first step is to obtain both geospatial boundaries and EHR data. The second step is to preprocess the EHR data to remove empty and erroneous values. The final step is to transform the data into a format that is suitable for cartograms. Geospatial boundaries, or shapefiles, were obtained from the sources described here.

3.1. Choropleth Shapefile

Clinical Commissioning Groups (CCGs) are the primary administrative and geographic unit of the National Health Service (NHS) in the UK [?]. The number of CCGs changes over time due to NHS reorganization. The most up-to-date shapefile is available from the Open Geography Portalx [?]. We decided to use the CCG shapefile from 2020 at the time of writing, due to the absence of published public EHR data based on the latest CCG reorganizations that took place in 2021 and 2022.

3.2. River Shapefiles

We used OpenStreetMap [?] as our data source to obtain shapefiles for the Thames River, the Trent River, and the Great Ouse River in England. These rivers were chosen as they are well-known rivers, pass through regions with dense populations, and provide informative geographical and topological cues. Although including smaller rivers is technically feasible, it may not increase the legibility of the cartogram. This is an open question for future work.

We first obtain a relation ID by searching for a river, e.g. River Thames, on OpenStreetMap. The relation ID is used to

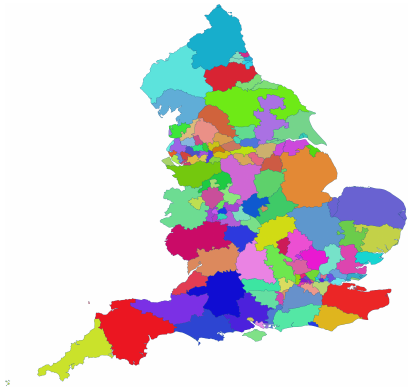


Figure 2: A map of 135 CCGs in England as of 2020, obtained from the Open Geography Portalx [?] with EPSG:4326 (WGS84 - World Geodetic System) as the Coordinate Reference System (CRS).

construct a query (See **??**) which enables the user to download the entire river shapefile using Overpass Turbo [**?**].

After acquiring the shapefiles, we used QGIS [**?**] to manually adjust the projections and convert them into GeoJSON files. Finally, mapshaper [**?**] is used to merge and convert the GeoJSON files into a TopoJSON file [**?**]. TopoJSON eliminates redundant coordinates in the data, improving the rendering speed of our implementation. See **??** on page **??** for the preprocessing result. We describe the one-time preprocessing steps in more detail in **??**.

3.3. EHR Data

We obtained the Clinical Commissioning Group Outcomes Indicator Set (CCG OIS) from NHS Digital [**?**]. The OIS is a set of indicators that are used to measure the quality of care and the associated health outcomes in the NHS. Some datasets include:

- Under 75 mortality: cardiovascular disease, respiratory disease, liver disease, and cancer
- Emergency hospital admission: stroke, alcohol-specific admission and readmission, coronary heart disease, readmissions within 30 days of discharge, children with lower respiratory tract infections

For all datasets, a spreadsheet including the following is provided:

- Reporting period: Calendar year of registration
- Period of coverage: Start and end date or reporting period
- Breakdown: Organization type
- ONS code: UK Office for National Statistics CCG code
- Level: CCG Code
- Level description: CCG Name
- Gender
- Indicator value: Directly standardized mortality rate
- CI lower: lower 95% confidence interval
- CI upper: upper 95% confidence interval
- Denominator: The count of registered patients
- Numerator: Number of deaths

Each CCG has a unique ONS code, which is used to link the CCG shapefile with the statistical data.

4. Demers Cartogram with Rivers

?? and **??** provide an overview of the hybrid cartogram layout process that includes rivers.

4.1. Initialization with Rivers

We first load and (optionally) render the CCG geospatial boundaries. For each CCG we compute the centroid and represent it using a square node, $\boldsymbol{n}$, with the initial size, s =1 pixel. We then load the river shapefiles and render the rivers. Since the vertices of the river in the shapefiles are not in sequential order, we first render the starting vertex, followed by the next nearest vertex. We do not need the original river resolution to incorporate them into cartograms. We reduce their resolution to match that of the cartogram nodes in order to facilitate node-river intersection tests. This rendering approach enables us to adjust the river resolution as shown in **??**. We further apply simplification by removing vertices that are too close to each other. The initialization procedure is a one-time process that can be saved for reuse.

4.2. Node Layout and Overlap Removal

We first apply the Fast Node Overlap Removal (FNOR) algorithm that solves the Variable Placement with Separation Constraints (VPSC) problem [**?**] in order to remove overlap between square nodes. We chose FNOR over other node overlap removal algorithms because FNOR is capable of minimizing spread and node movement while maintaining a good level of global shape preservation [**?**]. Initialized with a pixel size of unity, we gradually increase the node size by one unit at a time to ensure smooth transitions. An increase in s can cause the nodes to overlap. During overlap removal, we compute node trajectories (See **??**) and translate nodes to their new position. Nodes that cross a river, denoted $\boldsymbol{n}_{n_x r}$, are translated back to their previous position. If a node oscillates across a river, we identify this as a stalemate. One iteration of the layout ends when 1) no node overlap is present; AND 2) no nodes cross a river. We then increase s by one unit and repeat the algorithm until the average cartographic error, ϵ_c, a measure set by the user, reaches its maximum value $\epsilon_{c_{max}}$. The gradual size increase process provides stability to the layout, as can be seen in the accompanying video.

4.3. River Intersection Testing

The logic for translating the position of a node is detailed in **??**. When a node's position changes, we test if the node's trajectory intersects any segment of a river. See **??**. A bounding box intersection test between the edge defined by node translation and river edges can be performed to reduce the number of edge intersection tests required. Using the intersection test, we identify all nodes that cross the river as a result of the initial FNOR algorithm. We label these nodes, $\boldsymbol{n}_{n_x r}$.

4.4. Translating Rivers

For all nodes $\boldsymbol{n}_{n_x r}$ that cross a river, $\boldsymbol{r}$, we compute an average vector $\boldsymbol{v}_{avg}$ used to translate $\boldsymbol{r}$. Whenever a node, $\boldsymbol{n}$, crosses a river, we store a vector $\overrightarrow{\boldsymbol{n}\boldsymbol{n}_t}$ that points in the direction of the translation. We then use a heuristic to translate $\boldsymbol{r}$ using the average vector of node intersection

$$\boldsymbol{v}_{avg} = \sum_{i=1}^{\epsilon_t} \frac{\overrightarrow{\boldsymbol{n}_i \boldsymbol{n}_{i_t}}}{\epsilon_t}$$

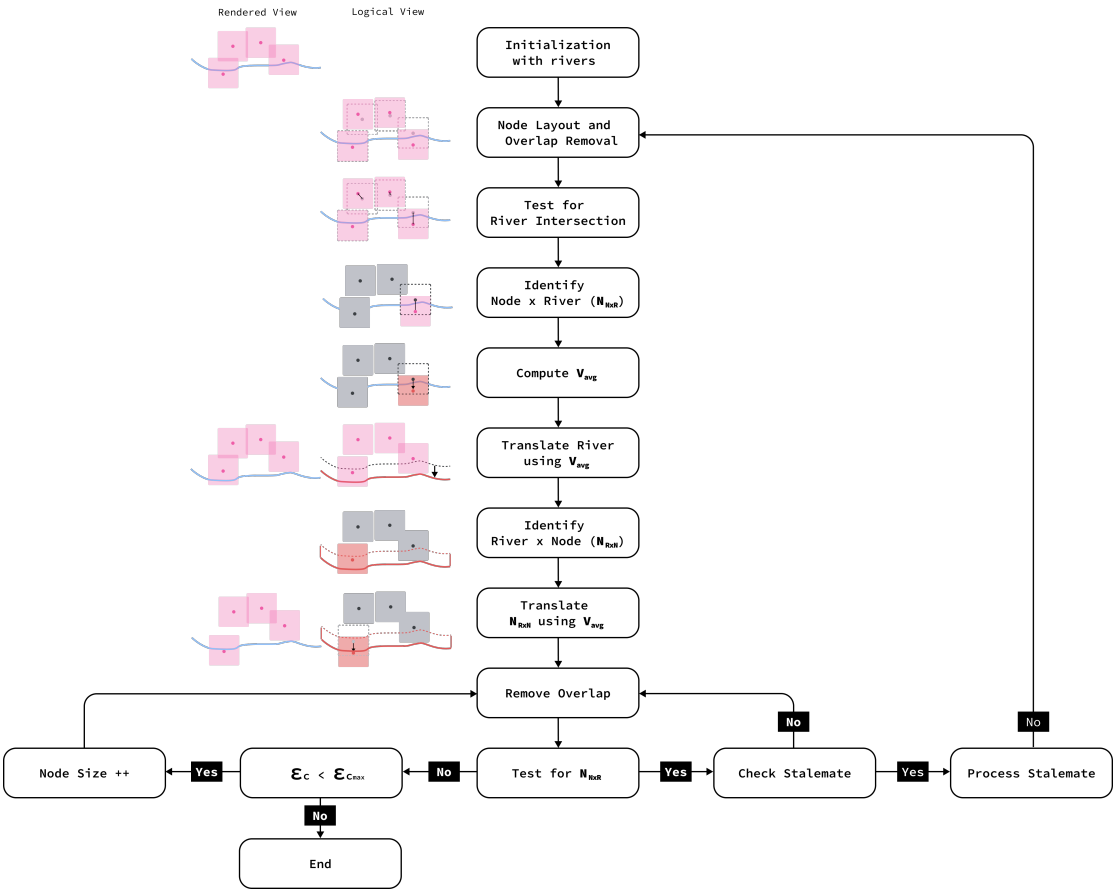


Figure 3: An overview of our hybrid layout algorithm incorporating rivers. See also **??** in Section **??** for more detail. See **??** for the logic of processing a stalemate. For illustration purposes, we show the rendered views alongside the logical views representing the actual computation.

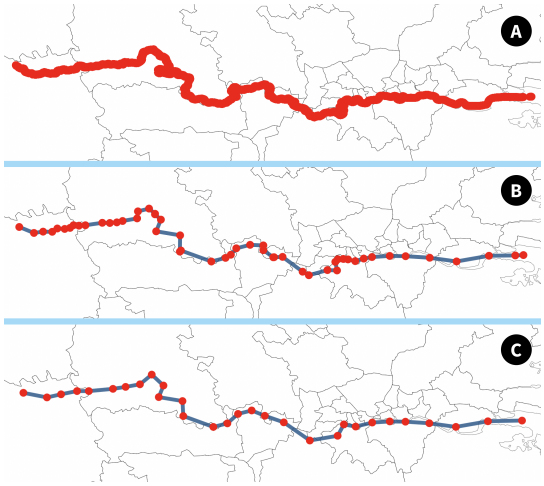


Figure 4: The resolution of rivers can be dynamically adjusted by the user. (A) shows River Thames at its original resolution with 10,170 edges. (B) shows the river at a reduced resolution of 49 edges. We further smooth the river by removing 19 vertices in dense areas, as shown in (C). The reduced resolution preserves the majority of River Thames' original shape and improves the performance of our river intersection tests.

, where ϵ_t is the number of nodes intersecting the river. This step intends to create space for the next iteration of node translation without crossing a river. The detailed procedure for translating rivers is provided by **??**.

When a river, $\boldsymbol{r}$, is translated by $\boldsymbol{v}_{avg}$, this can trigger a scenario where nodes are crossed by a translated river, denoted $\boldsymbol{n}_{r_xn}$. As a heuristic, we also translate these nodes by $\boldsymbol{v}_{avg}$. The reasoning behind this is that $\boldsymbol{v}_{avg}$ indicates which direction the river needs to be translated to create space for the nodes that are too crowded together. In practice, $\boldsymbol{v}_{avg}$ is multiplied by a scaling factor $\propto \boldsymbol{v}_{avg}$. Thus, we can influence how far $\boldsymbol{r}$ is translated in each iteration of the layout algorithm. We can use this to ensure smooth transitions between iterations.

4.5. Process Stalemates

As the FNOR always attempts to produce an optimal node layout where node distribution and translation are minimized, a node's translation path can repeatedly intersect a river due to congestion, creating a stalemate situation, as shown in **??**. If a node is translated between two positions, $\boldsymbol{n}$ and $\boldsymbol{n}_t$, for w iterations (a user-adjustable parameter), we introduce a heuristic solution: constructing a corridor to alleviate congestion. A corridor, c, is a rectangle with a width of c_w and a length of c_l, formed by deriving two edges $\overline{e_{p^1}}$ and $\overline{e_{p^2}}$ such that $\overline{e_{p^1}} \parallel \overline{e_{p^2}} \parallel \overline{\boldsymbol{n}_t\boldsymbol{n}_c}$ (See **??**C and D). All nodes enclosed by c are then translated by $\overrightarrow{\boldsymbol{n}_t\boldsymbol{n}}$ to alleviate the congestion (See **??**E). The procedure for constructing corridors is provided by **??**.

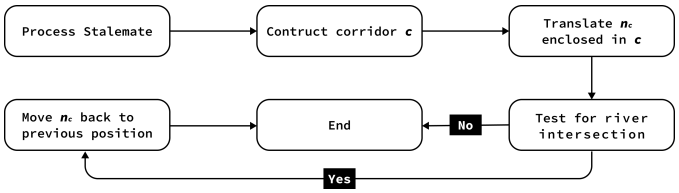


Figure 5: A flowchart illustration of stalemate processing. See **??**, **??**, and **??** for more detail.

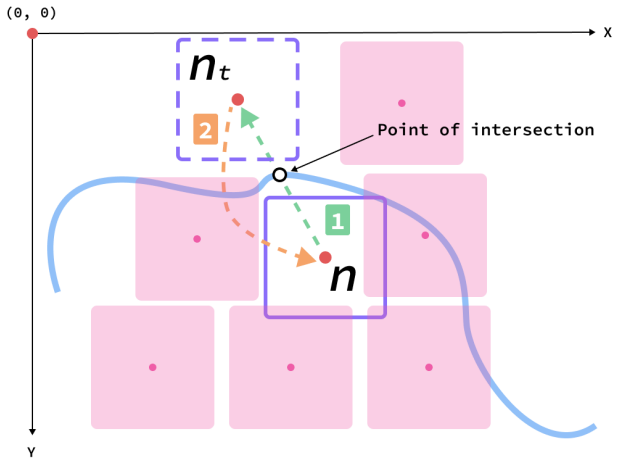


Figure 6: A stalemate: when a node's translation path $\overrightarrow{nn_t}$ (iteration 1) intersects a river w times. The node is translated back to its previous position (iteration 2). A stalemate can occur when the area is congested and the node cannot translate to a new position without intersecting a river.

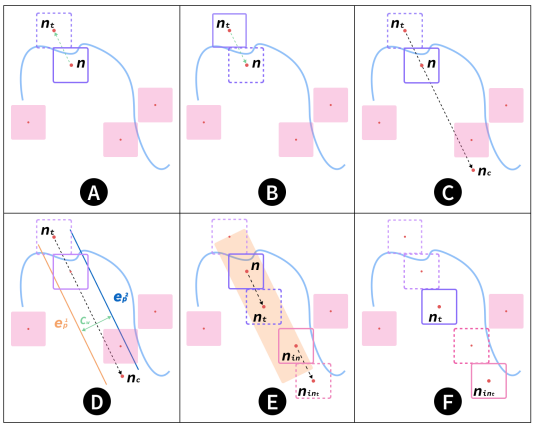


Figure 7: A stalemate occurs when a node's translation path $\overrightarrow{nn_t}$ intersects a river for w times, as shown in (A). To address this, we derive a corridor (orange rectangle in (E)) based on $\boldsymbol{n}$ and $\boldsymbol{n}_t$. All nodes within the corridor are translated based on $\overrightarrow{n_t n}$, such that $\overrightarrow{nn_t} = \overrightarrow{n_{in} n_{in_t}}$. For clarity in the illustration, we place nodes sparsely in this figure.

4.6. Terminating the Algorithm

The algorithm terminates when ϵ_c reaches $\epsilon_{c_{max}}$, the error tolerance set by the user. We adopt the maximum cartographic error from **?**], namely:

$$\epsilon_{c_{max}} = \max_{n \in L} \frac{|\boldsymbol{n}_i - \boldsymbol{n}_{i_t}|}{\max\{\boldsymbol{n}_i, \boldsymbol{n}_{i_t}\}}$$

where $\boldsymbol{n}_i$ and $\boldsymbol{n}_{i_t}$ are the initial and translated regions in the cartogram, $\boldsymbol{L}$ represent the list of regions, and $\epsilon_{c_{max}}$ is a normalized value that we express as a percentage. For a detailed derivation of the formula, refer to the work by **?**]. Because the algorithm processes node-river intersections, we can measure a novel kind of error, namely, topological error ϵ_t. We maintain $\epsilon_t = 0$, however, we can count how many nodes would have crossed a river if we did not test for this and simply let nodes cross rivers. We express ϵ_t in the normalized range $\epsilon_t \in [0, 1]$, where $\epsilon_{t_{max}}$ is the case where all nodes cross a river.

When the algorithm terminates, the node layout is considered optimal where no nodes have crossed or crossed a river (denoted $\epsilon_t = 0$ and $\epsilon_c < \epsilon_{c_{max}}$). Every node remains on the same side of the river as its centroid.

4.7. User Options

?? presents an overview of the application, including user options. The user can adjust the following parameters:

Rendering Visibility: The rendering visibility of various elements, including the choropleth shapefile, rivers, nodes, and node centroids, can be toggled on and off.

Node Mapping: Both size and color of the nodes can be mapped to different EHR attributes or set to uniform.

Overlap Removal Speed: The overlap removal process can be observed in a step-by-step manner, or the algorithm can be run automatically.

Maximum Cartographic Error: The user can adjust the maximum cartographic error, $\epsilon_{c_{max}}$, which is used to terminate the algorithm.

River Translation: The behavior of rivers during the process can also be adjusted by the user: 1) Enable river crossing: this option allows nodes to cross rivers. Nodes cannot cross rivers by default; 2) Disable river translation: this option disables the translation of rivers, rivers are translated by default. Both options are useful for generating different layouts and to observe the behavior of the hybrid layout algorithm.

Corridor Length: The user can define the length of a corridor that is used to resolve stalemate situations. A longer corridor length allows more nodes to be translated when a stalemate occurs, but may result in a less accurate layout. The default corridor length is three times the max node size.

River Thickness and Resolution: Increasing the thickness of rivers may improve the recognizability of the cartogram. Similarly, increasing the resolution of rivers, at the expense of the speed of node-river intersection test, may produce a layout with higher legibility.

5. User-Centered Evaluation

We conduct a within-subject user study to evaluate the effectiveness of our approach. We chose this type of evaluation

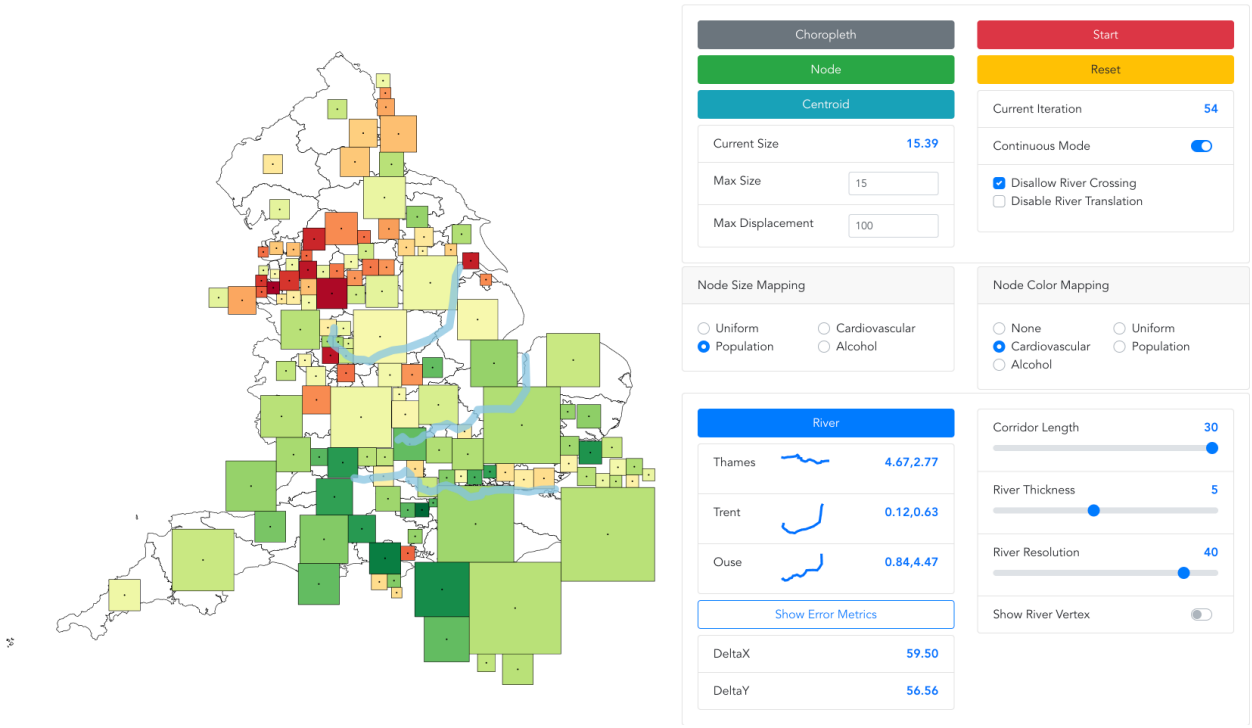


Figure 8: A screenshot of the user interface. User options are provided to adjust the terminating conditions (size and error), color mapping, visibility of nodes and rivers. Other options include the ability to control the overlap removal behavior: rivers can be static or dynamic. See **??** for more on user options. In this figure, $\epsilon_{c_{max}} = 1.875\%$, and an ϵ_t of 9.63% is eliminated.

because legibility is a human-centred characteristic. Many different types of statistical error metrics have been evaluated in previous work [**?**], however, our focus is more user-centric in nature.

5.1. Study Hypothesis

We formulate the following hypotheses to motivate our user study:

H1: The introduction of rivers can improve the legibility and recognizability of cartograms.

H1.1: The introduction of rivers can increase the accuracy of the target location.

H1.2: The introduction of rivers can reduce the time required to locate a target region in a cartogram.

We test these hypotheses using location-based tasks. Given a standard choropleth on the left side of the screen and the corresponding cartogram on the right side, we ask user study participants to locate the given region in the choropleth (on the left) in the cartogram (on the right). See **??**.

5.2. User Study Variables

In this section, we discuss the variables of our user study.

Independent Variables: The primary independent variable is the presence of rivers in the cartograms, which directly impacts the final layout of the cartogram. The identical set of rivers is either rendered or hidden collectively.

Dependent Variables: *Accuracy:* Given a CCG location in the choropleth on the left, we ask the participant to locate the corresponding node in the cartogram. Accuracy as the primary dependent variable is measured by the number of correct CCGs located and chosen by the participants. *Response time:* Another dependent variable is the time it takes the participants to complete each task.

Control Variables: *Choice of color map:* We use D3's built-in interpolateRdYlGn color map, a color scheme of red, yellow, and green, to depict the data in our cartograms. *Communicating the target CCG to the user:* We inform the participant about the target CCG they need to find. The target CCG is shown in the form of a nonstop blinking (between its original color and black) area on the screen, each blink is given a duration of 2 seconds.

5.3. User Study Design

In this section we describe the user study participants, datasets, and the experimental procedure. The study was approved by the University of Nottingham's Research Ethics Committee (Ref: 2021-2022-001).

Participants: We recruited 24 participants.

- Gender: 10 females and 14 males
- Age Group: 18-24 (16), 25-29 (8)
- Education: 1 PhD, 11 Master's, 8 Bachelor's, 4 Others

Datasets: We used the following EHR datasets for our evaluation:

- Population
- Under 75 mortality from cardiovascular disease

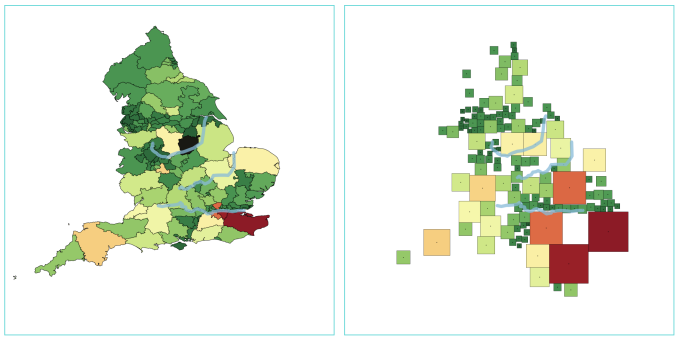


Figure 9: A sample location task for participants. The left shows the choropleth map, and the right shows the corresponding cartogram. Both images show the three longest rivers in England, and color is mapped to CCG population. The target CCG blinks on the choropleth (shown in black), and participants are asked to identify this CCG on the cartogram. In this figure, $\epsilon_{c_{max}} = 1.875\%$, and an ϵ_t of 6.67% is eliminated.

- Emergency admissions for alcohol-related liver disease
- Alcohol-specific admission and readmission

135 CCGs are rendered on the screen as a choropleth on the left. We then generated another view on the right using cartograms. The color of both visual designs is mapped to a data dimension. No cartograms were used more than once per participant. See **??** for an example.

5.4. Procedure

Due to pandemic restrictions, we designed the user study to be carried out online. Participants are expected to use their retail hardware for the tasks. However, devices with small screens are not recommended because of the large size of the cartograms that are rendered. The within-subject user study includes four parts:

P1: The participants are asked to listen to instructions and training provided as both text and videos. The instructions are designed to help participants understand the concepts used in the tasks. An instructional video was provided [1].

P2: The participants are given three practice tasks to familiarize themselves with the user study design. A sample task [2] is shown in **??** and included in the online materials[3]. The practice tasks are identital to the actual tasks, accompanied by instructions provided before the actual tasks commence. The results from these practice tasks are not included in the final analysis. These tasks also serve as a quality test to see whether users are taking the study seriously. A demonstration of three sample tasks is also included in the instructional video[1].

P3: The participants are asked to complete 16 location tasks that involve 4 target CCGs. See the online materials for exact locations of target CCGs[3]. Response and reaction time are recorded. These 4 CCGs are selected to avoid extreme cases (thus bias the result), in terms of size, color, and location.

[1]Instructional video: `https://www.youtube.com/playlist?list=PLL7sHvxLtD75fMtrUQrAdddjt3wfFkcWz`

[2]A sample task is accessible at `https://ghr.wangqiru.com/#/P1`.

P4: The participants are asked to complete a questionnaire[3] that consists of 5-Point Likert Scale questions and open-ended questions.

5.5. User Study Results

In this section, we delve into the analysis of the preliminary user study conducted. Our study involved the recruitment of 24 participants, each tasked with completing 16 location-related tasks. As a result, we collected a total of 384 responses. However, we excluded 24 responses from our analysis as they took over 60 seconds to complete. We hypothesize that these prolonged response times indicate potential distractions experienced by participants during the study.

Accuracy: ?? shows the accuracy measured by the number of correct CCGs chosen by the participants. The chart shows that the introduction of rivers has improved the accuracy of two CCGs. The accuracy for the two groups differs significantly according to the two-sample unequal variance t-test performed, $t(262.98) = 3.76$, $p = 0.0002$. This supports our hypothesis H1.1 that the introduction of rivers increases target location accuracy.

Response Time: ?? shows the average time to locate a CCG measured for four CCGs. The chart indicates that the introduction of rivers has slightly reduced the time needed to locate CCGs. The two groups show no statistically significant differences according to the two-sample unequal variance t-test performed, $t(356.03) = 0.87$, $p = 0.38$. However, the average time to locate a given CCG is reduced for 3 out of the 4 CCGs users in the study.

Participant Feedback: ?? shows the results of the following Likert Scale questions:

(A) 87.5% of participants agree that including rivers is useful.

[3]The materials for P1-P4 are available on the Open Science Foundation website at `https://osf.io/q39w7`.

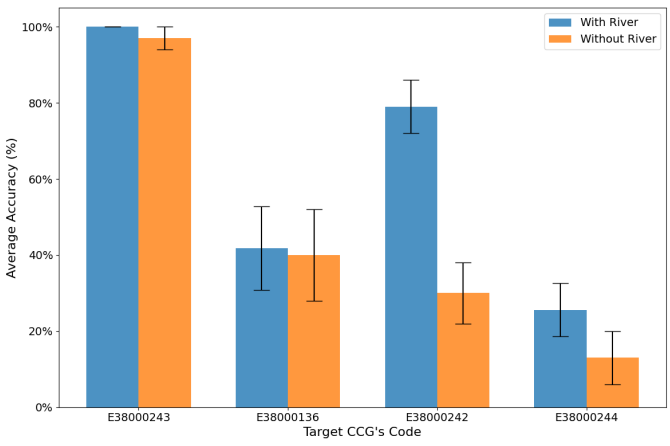


Figure 10: In the bar chart, the y-axis shows the average accuracy rate of locating CCGs with and without rivers for user study participants. The x-axis shows the four target CCGs, as described in Procedure P3.

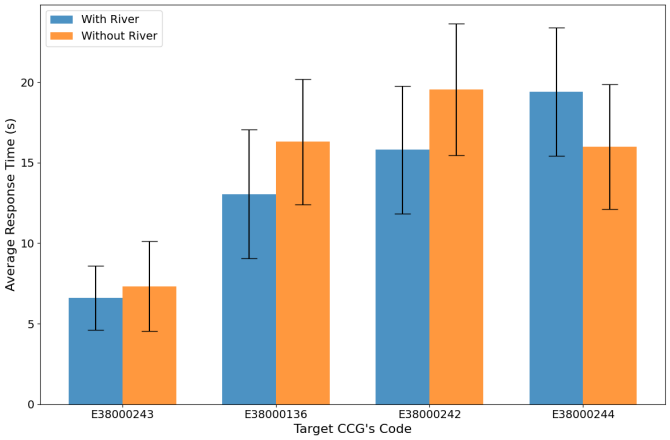


Figure 11: In the bar chart, the y-axis shows the average response time for locating CCGs with and without rivers for user study participants. The x-axis shows the four target CCGs, as described in Procedure P3.

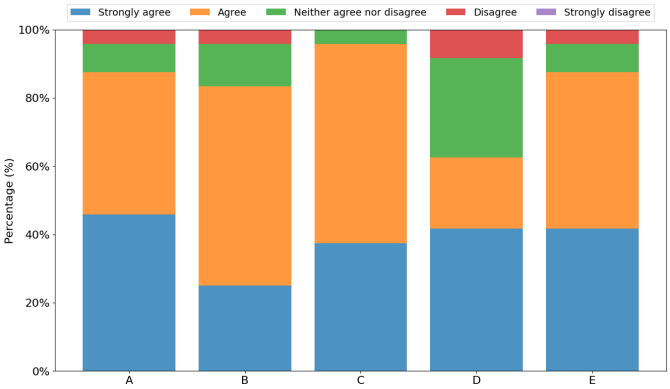


Figure 12: The stacked bar chart shows the user study participant responses of Likert Scale questions.

(B) 83.3% of participants agree that rivers increase the legibility of a cartogram.

(C) 95.8% of participants agree that including rivers makes cartograms easier to understand.

(D) 62.5% of participants agree that including rivers makes CCGs easier to locate.

(E) 87.5% of participants agree that including rivers adds value to the standard cartogram.

The qualitative results clearly support our hypothesis H1 that rivers generally increase the legibility of cartograms.

6. Limitations and Future Work

In this section, we discuss some limitations of our work and future research directions. Because this is a new concept, it opens the door for many future research directions.

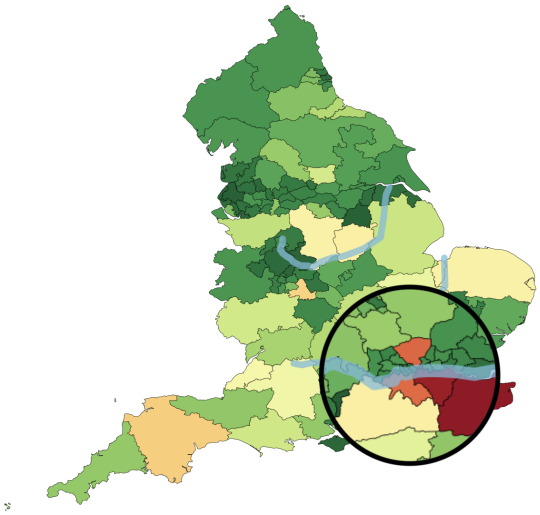


Figure 13: Due to color and relative location, we believe the CCGs in the black circle are easier to locate.

6.1. Color map choice

The first limitation is the color map that we use to depict the data in our cartograms. We use D3's built-in interpolateRdYlGn color map, a diverging color scheme of red, yellow, and green. However, we believe that the choice of color map can have a significant impact on the legibility of cartograms. In the user study, we carefully avoid extreme values where the location or color of the CCG makes it easier to locate the target. See **??** for an example. We plan to explore the impact of different color maps on the legibility of cartograms in future work.

6.2. Overlap removal algorithm choice

Another limitation is the algorithm (FNOR) we use. We believe that developing a new algorithm with built-in constraint support can significantly reduce the time required to generate cartograms with rivers. Currently, the runtime of our layout algorithm is approximately 30 milliseconds for each iteration. When the quantity of nodes and features increases, generating the optimal layout demands several hundred iterations.

6.3. Generalizability

Future work also includes generalizations and extensions of the algorithm, e.g., the use of other features in the cartogram layout such as additional rivers, major highways, lakes, and coastlines, etc. We also consider whether increasing the length of the rivers as the size of the nodes increases would be a useful option. We would like to explore the case of river-river intersections (or confluence) and testing out more geographies such as the U.S. and Europe, which have more complex rivers. We also considered the idea of deforming the rivers as part of the layout algorithm, however, this idea is open to future work.

6.4. Improved User Study

Due to pandemic restrictions, the user study did not take place in a controlled environment. In the future, we aim to execute a more controlled laboratory study, ensuring a controlled environment and standardized hardware for participants.

7. Conclusions

In this paper, we present a novel hybrid layout approach to generate non-contiguous cartograms with rivers. We first propose a new algorithm to generate cartograms with rivers, and then we present a prototype to support the exploration of cartograms with rivers. We evaluate our approach with a preliminary user study, and the results support our hypotheses: the introduction of rivers improves the legibility and recognizability of cartograms, although a deeper analysis is necessary to reach more confident conclusions. We also discuss the limitations of our work and future research directions.

Appendix A. Obtaining Shapefile

```
relation(2263653);>>;
// River Great Ouse: 2798097
// River Trent: 2863468
out skel;
```

Listing 1: The query that downloads the shapefile of River Thames from OpenStreetMap via the Overpass Turbo API.

Appendix B. Pre-processing Shapefiles

Shapefiles from different sources are likely to be incompatible. In our case, the NHS CCG shapefile is incompatible with the river shapefiles. The major reason for the incompatibility is the coordinate reference system (CRS). The CRS of the CCG shapefile is EPSG:27700 (OSGB36 - British National Grid). The CRS of the river shapefiles is EPSG:4326 (WGS84 - World Geodetic System). Here, we provide some pre-processing steps using QGIS (version: 3.26.0-Buenos Aires) [**?**] to handle the incompatibility and reduce shapefile size to improve performance.

Appendix B.1. Import Shapefiles into QGIS

We first load all three river shapefiles into QGIS **??**, followed by the CCG shapefile **??**.

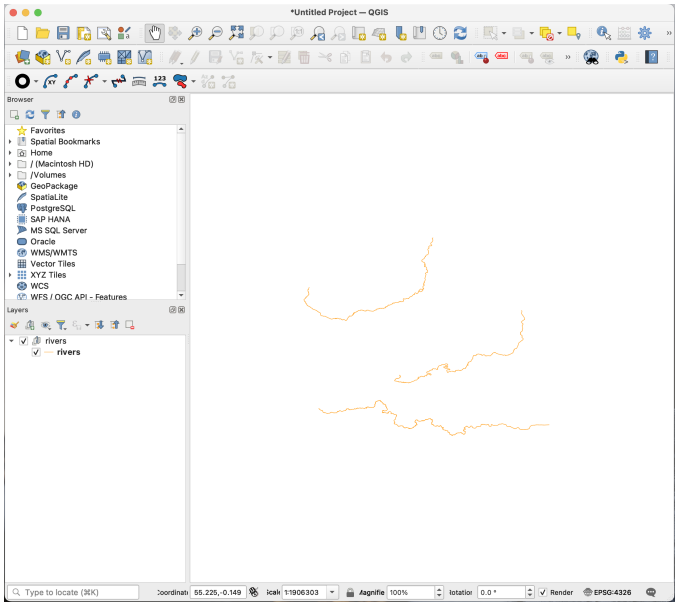


Figure B.14: QGIS interface, with River Trent, River Great Ouse, and River Thames (from top to bottom) imported.

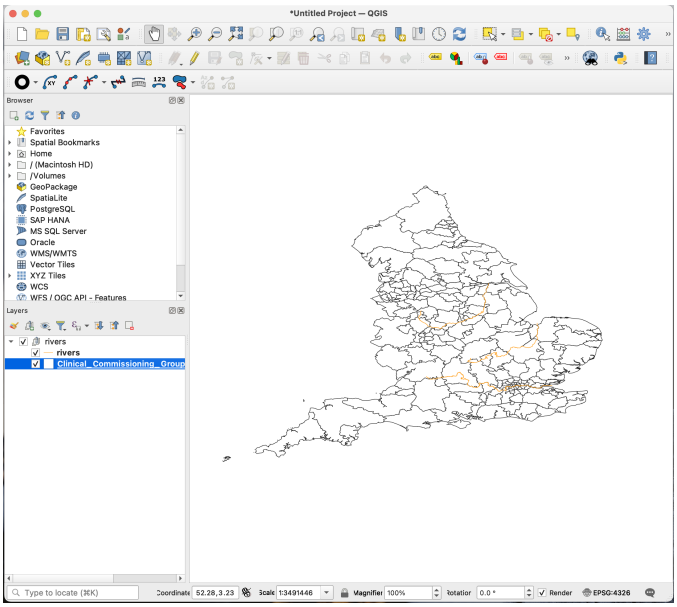


Figure B.15: QGIS interface, with all NHS CCGs imported.

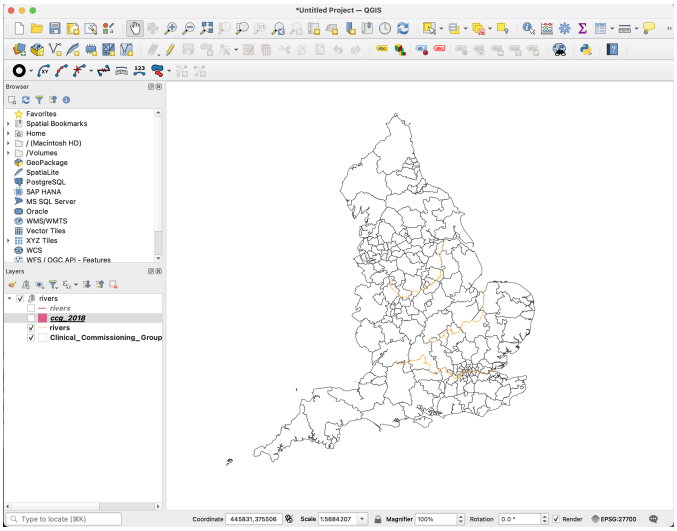


Figure B.16: QGIS interface, showing the unified CRS (OSGB36) for both layers.

Appendix B.2. Export Shapefiles in GeoJSON and Unify the Coordinate Reference System (CRS)

We then use QGIS to unify the CRS, and export both layers in GeoJSON format. See **??** and **??**. The unified layer is shown in **??**.

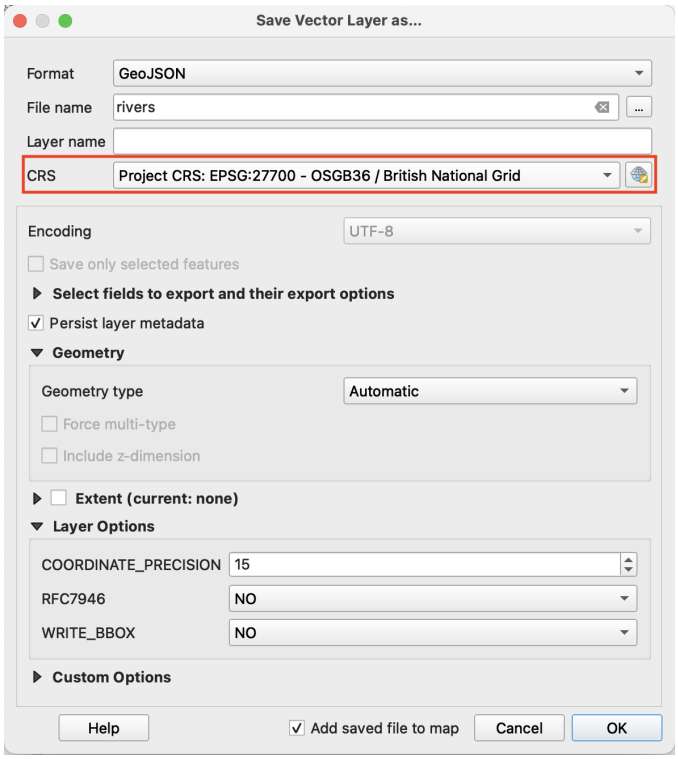


Figure B.17: QGIS interface, exporting all rivers using the OSGB36 CRS in GeoJSON.

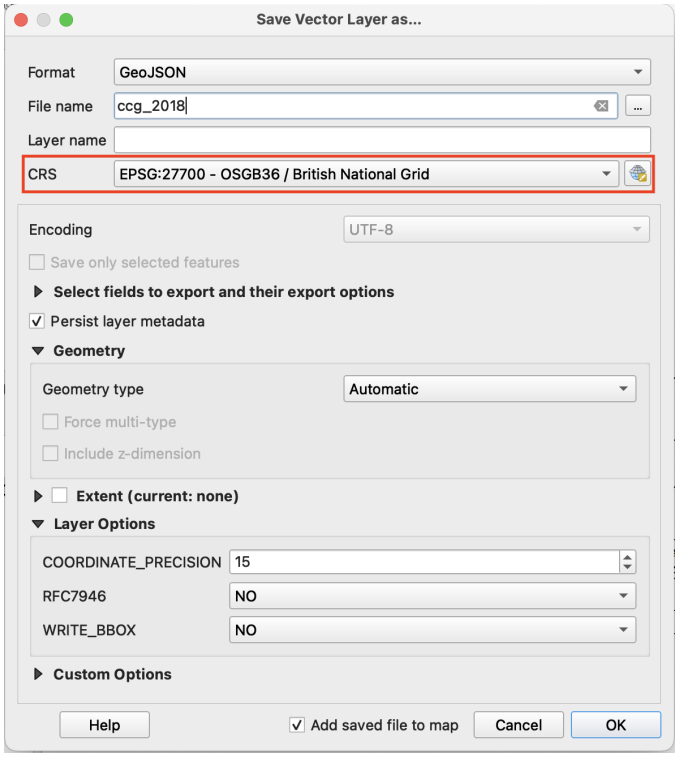


Figure B.18: QGIS interface, exporting all NHS CCGs using the OSGB36 CRS in GeoJSON.

Appendix B.3. Merge Shapefiles and Reduce File Size

We then merge two layers into one layer, and export it in the TopoJSON format using Mapshaper [**?**]. Mapshaper also supports the simplification of GeoJSON shapefiles. See **??**.

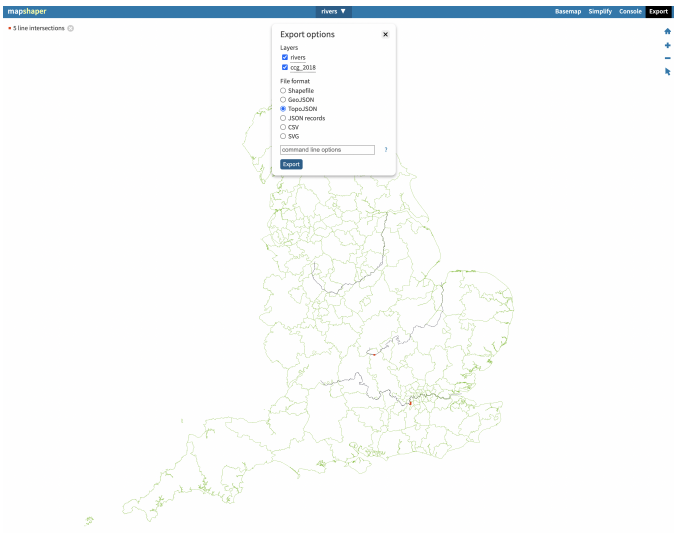


Figure B.19: Mapshaper interface, merging all rivers with NHS CCGs into one layer, and export the merged layer in TopoJSON.

Appendix B.4. Pre-processing Result

?? shows the preprocessing result. The reduction in file size is significant and greatly reduces the initialization time of our implementation.

Shapefile	Original	GeoJSON	TopoJSON
Rivers	2.0 MB (GeoJSON)	1.4 MB	-
NHS CCGs	46.6 MB (.shp, Esri vector shapefile)	140.2 MB	-
Merged	-	-	16.3 MB

Table B.3: The file size is reduced by 88.5% from the original size.

Appendix C. Procedure: DeriveParallelEdge

Algorithm 1 Procedure to derive an edge, $\overline{e_p}$, that is parallel to $\overline{e}$ with a distance of d.

Input:
$\overline{e} \leftarrow$ the edge used to derive the parallel edge $\overline{e_p}$
$d \leftarrow$ the shortest distance between $\overline{e}$ and $\overline{e_p}$

Output:
An edge, $\overline{e_p}$, that is parallel to $\overline{e}$ with a distance of d.

Local variables:
$\Delta x, \Delta y \leftarrow$ the differences in x, y for $\overline{e}.start$ and $\overline{e}.end$
$scale \leftarrow$ the scale of $\frac{d}{\sqrt{\Delta x^2+\Delta y^2}}$

1: **procedure** DeriveParallelEdge($\overline{e}$, d)
2: $\Delta x \leftarrow \overline{e}.start.x - \overline{e}.end.x$
3: $\Delta y \leftarrow \overline{e}.start.y - \overline{e}.end.y$
4: $scale \leftarrow \frac{d}{\sqrt{\Delta x^2+\Delta y^2}}$
5: $\overline{e_p}.start.x \leftarrow scale \cdot (-\Delta y) + \overline{e}.start.x$
6: $\overline{e_p}.start.y \leftarrow scale \cdot \Delta x + \overline{e}.start.y$
7: $\overline{e_p}.end.x \leftarrow scale \cdot (-\Delta y) + \overline{e}.end.x$
8: $\overline{e_p}.end.y \leftarrow scale \cdot \Delta x + \overline{e}.end.y$
9: **return** $\overline{e_p}$
10: **end procedure**

Appendix D. Procedure: TranslateNode

Algorithm 2 Procedure to translate node positions.

Input:
$\boldsymbol{L} \leftarrow$ a list of $\boldsymbol{n}$ representing regions

Output:
$\epsilon_t \leftarrow$ the number of nodes crossing the river in the input

Global variables:
$w \leftarrow$ the maximum number of iterations indicating a stalemate

Local variables:
$\boldsymbol{n} \leftarrow$ a node is an object with the following properties:
$\boldsymbol{n}.x, \boldsymbol{n}.y$, or $\boldsymbol{n}(x, y) \leftarrow$ the x and y coordinates of $\boldsymbol{n}$
$\boldsymbol{n}.cross \leftarrow$ the number of times that $\boldsymbol{n}$ crosses a river
$\boldsymbol{n}_p \leftarrow$ the previous position of $\boldsymbol{n}$
$\boldsymbol{n}_t \leftarrow$ the translated position of $\boldsymbol{n}$

1: **procedure** TranslateNode($\boldsymbol{L}$)
2: $\epsilon_t \leftarrow 0$
3: **for each** $\boldsymbol{n} \in \boldsymbol{L}$ **do**
4: **if** $\boldsymbol{n}(x, y) \neq \boldsymbol{n}_t(x, y)$ **then**
5: $\boldsymbol{n}(x, y) \leftarrow \boldsymbol{n}_t(x, y)$
6: **if** TestIntersection($\overline{\boldsymbol{n}\boldsymbol{n}_t}$) = $True$ **then**
7: $\boldsymbol{n}.cross++$
8: ϵ_t++
9: **if** $\boldsymbol{n}.cross < w$ **then**
10: ▷ Translate back to previous position
11: $\boldsymbol{n}(x, y) \leftarrow \boldsymbol{n}_p(x, y)$
12: **else**
13: ProcessStalemate($\boldsymbol{n}, \boldsymbol{n}_t$)
14: $\boldsymbol{n}.cross \leftarrow 0$ ▷ Reset counter
15: **end if**
16: **end if**
17: **end if**
18: **end for**
19: **return** ϵ_t
20: **end procedure**

Appendix E. Procedure: UpdateLayout

Algorithm 3 Procedure to adjust river positions, remove node overlap and prevent nodes from crossing rivers. See **??**.

Global variables:
$\boldsymbol{L}$ ← a list of $\boldsymbol{n}$ representing regions with the following properties:
  $\boldsymbol{L}.cross$ ← the number of $\boldsymbol{n}$ in $\boldsymbol{L}$ that crosses a river

s ← the initial size of all nodes
ϵ_c ← the average cartographic error of all nodes
$\epsilon_{c_{max}}$ ← the maximum cartographic error of all nodes
w ← the maximum number of iterations indicating a stalemate

Local variables:
$\boldsymbol{n}$ ← a node is an object with the following properties:
  $\boldsymbol{n}.x$, $\boldsymbol{n}.y$, or $\boldsymbol{n}(x, y)$ ← the x and y coordinates of $\boldsymbol{n}$
  $\boldsymbol{n}.cross$ ← the number of times that $\boldsymbol{n}$ crosses a river
  $\boldsymbol{n}_p$ ← the previous position of $\boldsymbol{n}$
  $\boldsymbol{n}_t$ ← the translated position of $\boldsymbol{n}$

```
1:  procedure UpdateLayout
2:      s ← 1
3:      while ε_c < ε_{c_max} do
4:          L.cross ← 1                          ▷ Trigger the while loop
5:          while L.cross > 0 do
6:              L.cross ← 0
7:              L ← RemoveOverlap (L)
8:              TranslateRiver(L)
9:              L.cross ← TranslateNode(L)
10:         end while
11:         s + +
12:     end while
13: end procedure
```

Appendix F. Procedure: TranslateRiver

Algorithm 4 Procedure to translate rivers.

Input:
$\boldsymbol{L}$ ← a list of $\boldsymbol{n}$ representing regions

Local variables:
$\boldsymbol{R}$ ← a list of $\boldsymbol{r}$ representing river features
$\boldsymbol{n}$ ← a node is an object with the following properties:
  $\boldsymbol{n}.x$, $\boldsymbol{n}.y$, or $\boldsymbol{n}(x, y)$ ← the x and y coordinates of $\boldsymbol{n}$
  $\boldsymbol{n}_p$ ← the previous position of $\boldsymbol{n}$
  $\boldsymbol{n}_t$ ← the translated position of $\boldsymbol{n}$
ϵ_t ← the number of nodes intersecting $\boldsymbol{r}$

```
1:  procedure TranslateRiver(L)
2:      for each r ∈ R do
3:          ε_t ← 0
4:          v_r ← (0, 0)                  ▷ Hold the sum of vectors nn_t
5:          for each n ∈ L do
6:              if n(x, y) ≠ n_t(x, y) then
7:                  if TestIntersection(nn_t, r) = True then
8:                      v_r ← v_r + nn_t
9:                      ε_t + +
10:                 end if
11:             end if
12:         end for
13:         Translate river r by the average vector v_r / ε_t
14:     end for
15: end procedure
```

Appendix G. Procedure: DerivePoint

Algorithm 5 Procedure to derive a point based an edge and a distance.

Input:
$\overline{e}$ ← the edge used to derive the new point
d ← the distance between $\boldsymbol{n}_c$ and $\overline{e}.start$

Output:
A point, $\boldsymbol{n}_c$, that is distance d away from $\overline{e}.start$.

Local variables:
$\Delta x, \Delta y$ ← the differences in x, y for $\overline{e}.start$ and $\overline{e}.end$

```
1:  procedure DerivePoint(e, d)
2:      Δx ← e.start.x − e.end.x
3:      Δy ← e.start.y − e.end.y
4:      n_c.x ← Δx / sqrt(Δx² + Δy²) · d
5:      n_c.y ← Δy / sqrt(Δx² + Δy²) · d
6:      return n_c
7:  end procedure
```

Appendix H. Procedure: ProcessStalemate

Algorithm 6 Procedure to derive a corridor to resolve stalemate. We use an SVG canvas, where the point of origin (0,0) is located at the top left corner, with the x-axis extending to the right and the y-axis extending downwards (See **??**).

Input:
$\boldsymbol{n} \leftarrow$ the node used to derive the corridor

Global variables:
$c_l \leftarrow$ the length of a corridor
$c_w \leftarrow$ the width of a corridor

Local variables:
$c \leftarrow$ the corridor
$\boldsymbol{n}_c \leftarrow$ the point extending $\overrightarrow{\boldsymbol{n}_t\boldsymbol{n}}$ such that $|\boldsymbol{n}_t\boldsymbol{n}_c| = c_l$
$\overline{e_{p^1}}, \overline{e_{p^2}} \leftarrow$ the edges parallel to $\overline{\boldsymbol{n}_t\boldsymbol{n}_c}$
corridor $\leftarrow$ a rectangle formed by $\overline{e_{p^1}}$ and $\overline{e_{p^2}}$

procedure ProcessStalemate($\boldsymbol{n}$)
  $\boldsymbol{n}(x, y) \leftarrow \boldsymbol{n}_p(x, y)$
  $\boldsymbol{n}_c \leftarrow$ DerivePoint($\overrightarrow{\boldsymbol{n}_t\boldsymbol{n}}$, c_l)
  $\overline{e_{p^1}} \leftarrow$ DeriveParallelEdge($\overrightarrow{\boldsymbol{n}_t\boldsymbol{n}_c}$, $\frac{c_w}{2}$)
  $\overline{e_{p^2}} \leftarrow$ DeriveParallelEdge($\overrightarrow{\boldsymbol{n}_t\boldsymbol{n}_c}$, $-\frac{c_w}{2}$)
  $c \leftarrow \begin{bmatrix} \overline{e_{p^1}}.start & \overline{e_{p^1}}.end \\ \overline{e_{p^2}}.start & \overline{e_{p^2}}.end \end{bmatrix}$
  for each $\boldsymbol{n}_{in}$ inside c **do**
    $\overrightarrow{\boldsymbol{n}_{in}\,\boldsymbol{n}_{in_t}} = \overrightarrow{\boldsymbol{n}_t\boldsymbol{n}}$
    $\boldsymbol{n}_{in}(x, y) \leftarrow \boldsymbol{n}_{in_t}(x, y)$
  end for
end procedure

Appendix I. Procedure: TestIntersection

Algorithm 7 Procedure to test if a node's translation path, $\overline{\boldsymbol{n}\boldsymbol{n}_t}$ intersects a river.

Input:
$\overline{\boldsymbol{n}\boldsymbol{n}_t} \leftarrow$ the node's translation path
$\boldsymbol{r} \leftarrow$ a river feature

Output:
Returns *True* if the node crosses a river.

Local variables:
$b_{\overline{n}}, b_{\overline{e}} \leftarrow$ the bounding boxes for $\overline{\boldsymbol{n}\boldsymbol{n}_t}$ and $\overline{e}$
$\overline{e} \leftarrow$ an edge of $\boldsymbol{r}$

procedure TestIntersection($\overline{\boldsymbol{n}\boldsymbol{n}_t}$, $\boldsymbol{r}$)
  for each $\overline{e} \in \boldsymbol{r}$ **do**
    $b_{\overline{n}} \leftarrow$ GetBoundingBox ($\overline{\boldsymbol{n}\boldsymbol{n}_t}$)
    $b_{\overline{e}} \leftarrow$ GetBoundingBox ($\overline{e}$)
    if $b_{\overline{n}}$ *intersect* $b_{\overline{e}} = True$ **then**
      return $\overline{\boldsymbol{n}\boldsymbol{n}_t}$ *intersect* $\overline{e}$
    end if
  end for
  return *False*
end procedure