Journal of Statistical Software

August 2018, Volume 86, Code Snippet 1. *doi: 10.18637/jss.v086.c01*

Rectangular Statistical Cartograms in R: The recmap Package

Christian Panse
Swiss Federal Institute of Technology Zurich

Abstract

Cartogram drawing is a technique for showing geography-related statistical information, such as demographic and epidemiological data. The idea is to distort a map by resizing its regions according to a statistical parameter by keeping the map recognizable. This article describes an R package implementing an algorithm called RecMap which approximates every map region by a rectangle where the area corresponds to the given statistical value (maintain zero cartographic error). The package implements the computationally intensive tasks in C++. This paper's contribution is that it demonstrates on real and synthetic maps how package **recmap** can be used, how it is implemented and how it is used with other statistical packages.

Keywords: cartogram, spatial data analysis, geovisualization, demographics, R.

1. Introduction

The idea of generating a cartogram is to distort a map by resizing its regions according to a given statistical parameter, but in a way that keeps the map recognizable. These so-called cartograms or *value-by-area maps* may be used to visualize any geo-spatial related data, e.g., political, economic, or public health data. There exist several algorithms to compute so-called contiguous cartograms. An overview on historical, hand-drawn, and computer generated cartograms can be found in Tobler (2004) and Nusrat and Kobourov (2016).

For using *contiguous cartograms*, the diffusion-based method of Gastner and Newman (2004) is available through the R packages **Rcartogram** and **getcartr** (Temple Lang 2016; Brunsdon and Charlton 2014).

An alternative approach to *contiguous cartograms* is to entirely relax the map topology by approximating each map region by basic geometric objects like rectangles or circles (Dorling 1996). Such rectangular cartograms can be generated from geolocation and statistical data.

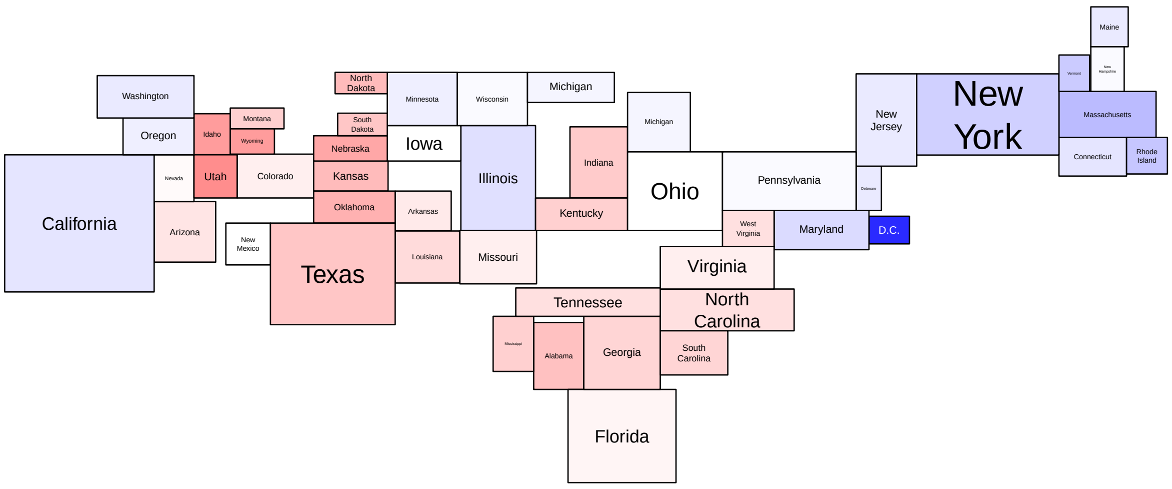

Figure 1: A rectangular statistical cartogram of the US election in 2004 is drawn. The area corresponds to the number of electors. Color is indicating the outcome of the vote. Democrats are represented by the color blue and Republicans are represented through red coloring. Regions with low saturation, e.g., Ohio, Pennsylvania, and Florida, highlight states with a tight outcome of the vote (also known as swing states). The election cartogram was computed by using the original implementation of the construction heuristic RecMap MP2 introduced by Heilmann *et al.* (2004). Map source: US Census Bureau; election data source: `http://www.electoral-vote.com/`, November 2004.

Hence, they provide a useful alternative, even if there are no boundaries available or some statistical values are missing. First rectangular cartograms were drawn by hand following a system of construction (Raisz 1934). Recent research publications on rectangular cartogram drawing include Van Kreveld and Speckmann (2004, 2007); Buchin, Speckmann, and Verdonschot (2012) and Buchin, Eppstein, Löffler, Nöllenburg, and Silveira (2016). However, according to a recent publication, both variants of RecMap (Heilmann, Keim, Panse, and Sips 2004) are the only rectangular cartogram algorithms that "maintain zero cartographic error" (Nusrat and Kobourov 2016, section 5.4).

The R (R Core Team 2018) package **recmap** (Panse 2018) discussed in this article contains an implementation of the RecMap (Map Partition variant 2) algorithm (Heilmann *et al.* 2004) to draw maps according to given statistical parameter. A typical usage of a cartogram based visualization is demonstrated in Figure 1. Package **recmap** is available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=recmap`.

The article is organized as follows: The next section defines the input and output of the RecMap algorithm and the objective functions. In Section 3, the usage of the **recmap** package using the R shell is demonstrated. Section 4 discusses major implementation details and provides some benchmark and performance studies. Section 5 describes how two metaheuristics can be used to find an optimized cartogram drawing. In Section 6, some applications are presented. Section 7 summarizes and discuss the approach.

2. Problem definition and objective functions

The input consists of a map represented by overlapping rectangles $\mathcal{R} = (r_1, \ldots, r_n)$. Each map region r_j contains:

- a tuple of (x, y) values corresponding to the (longitude, latitude) position,
- a tuple of (dx, dy) of expansion along (longitude, latitude),
- and a statistical value z.

The (x, y) coordinates represent the center of the minimal bounding boxes (MBBs). The coordinates of the MBB are derived by adding or subtracting the (dx, dy) tuple accordingly. A tuple (dx, dy) also defines the ratio of the corresponding map region. The statistical values $(z_1, \ldots, z_n)$ determine the desired area of each map region.
The ordering Π is an index vector taken from the permutation set $\mathrm{Perm}(n)$.

The output is a rectangular cartogram $\overline{\mathcal{R}}$ where the map regions are:

- non-overlapping,
- connected,
- rectangles are placed parallel to the axes.

Furthermore, for each map region the following criteria have to be satisfied:

- the area is equal to the desired area derived from the as input given statistical value z,
- the ratio, dy/dx, is preserved.

The `recmap` construction heuristic is a function

$$f : \mathbb{R}^{n\times 2} \times \mathbb{R}_{>0}^{n\times 3} \times \mathrm{Perm}(n) \quad \rightarrow \quad \mathbb{R}^{n\times 2} \times \mathbb{R}_{>0}^{n\times 2}, \tag{1}$$

which transforms the set of input rectangles $\mathcal{R}$ and a permutation Π into a rectangular cartogram

$$\overline{\mathcal{R}} \quad = \quad f(\mathcal{R}, \Pi), \tag{2}$$

so that important spatial constraints, in particular

- the topology of the dual graph $G(\mathcal{R}, E)$, defined by the overlapping input rectangles,
- the relative position of map region centers,

are tried to be preserved.

If the output satisfies these criteria, the rectangular cartogram is denoted as a *feasible solution*.

The following equations were introduced by Keim, North, and Panse (2004, Definition 1). The desired area $\tilde{A}_j$ of a map region r_j is defined as

$$\tilde{A}_j \quad = \quad z_j \cdot \frac{\sum_{i=1}^n A(r_i)}{\sum_{i=1}^n z_i}, \tag{3}$$

where the area of the rectangle r is defined by

$$A(r) \quad = \quad 4 \cdot dx \cdot dy. \tag{4}$$

The objective functions for area d_A, shape d_S (ratios of the MBBs), relative position d_R, and map topology d_T, are as defined and described by Heilmann *et al.* (2004, Equations 2–4):

$$\begin{aligned}
d_A &= d_A(\mathcal{R}, \overline{\mathcal{R}}) && (5)\\
&= \sum_{j=1}^{n} |A_j - \tilde{A}_j| && (6)\\
d_S &= d_S(\mathcal{R}, \overline{\mathcal{R}}) && (7)\\
&= \sum_{j=1}^{n} |(dy_j/dx_j) - (\overline{dy_j}/\overline{dx_j})| && (8)\\
d_T &= d_T(\mathcal{R}, \overline{\mathcal{R}}) && (9)\\
&= \frac{|\overline{E}_a \backslash E_a| + |E_a \backslash \overline{E}_a|}{|\overline{E}_a \cup E_a|}, && (10)\\
d_R &= d_R(\mathcal{R}, \overline{\mathcal{R}}) && (11)\\
&= \frac{2}{n \cdot (n-1)} \cdot \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} |\measuredangle\underbrace{(r_i, r_j)}_{\in \mathcal{R}} - \measuredangle\underbrace{(\tilde{r_i}, \tilde{r_j})}_{\in \overline{\mathcal{R}}}| && (12)
\end{aligned}$$

Note, if $\forall j \in \{1, \ldots, n\} : A_j = \tilde{A}_j \Rightarrow d_A = 0$ and if $\forall j \in \{1, \ldots, n\} : dy_j/dx_j = \overline{dy_j}/\overline{dx_j} \Rightarrow d_S = 0$. $n = |\mathcal{R}|$ and E denotes the edges of the dual graph. dT determines the differences in the pseudo dual graphs. dR measures the angle differences between all pairs of rectangle centers of the input map and output cartogram by using $\measuredangle(x, y)$ defined as follows[1]:

$$\measuredangle(x, y) \quad = \quad \arctan_2(x, y) \tag{13}$$

To find an optimal rectangular statistical cartogram out of all feasible solutions, as it will be intended in this manuscript using **recmap**, the optimization problem can be expressed as follows:

$$\begin{aligned}
&\underset{\Pi \in \text{Perm}(n)}{\text{minimize}} && a \cdot d_R + b \cdot d_T \text{ with } a, b \in \mathbb{R}_{\geq 0} && (14)\\
&\text{subject to} && d_A = 0, d_S = 0. && (15)
\end{aligned}$$

To find a sufficiently good solution for the optimization problem a metaheuristic, as described in Section 5, will be applied.

[1]Implemented using the C++ method `std::atan2` – "Computes the arc tangent of y/x using the signs of arguments to determine the correct quadrant." `http://en.cppreference.com/w/cpp/numeric/math/atan2`, access: 2017-01-01.

Term	Description	Equation
$\mathcal{R} = (r_1, \ldots, r_n)$	overlapping input rectangles	
(x, y)	coordinates represent the center of a rectangle	
(dx, dy)	expansion along x and y axes	
z	statistical value of a rectangle	
$G(\mathcal{R}, E)$	dual graph of $\mathcal{R}$	
Π	permutation	
$\tilde{A}_j$	desired area of a map region j	3
$A(r)$	area of a rectangle	4
d_A	area error	5
d_S	shape error	8
d_T	topology error	10
d_R	relative position error	12
$\measuredangle(x, y)$	angle between two points x and y in $\mathbb{R}^2$	13
f	construction heuristic	1
$\overline{\mathcal{R}}$	output / cartogram	2

Table 1: The table provides a glossary of the used algebraic terms.

3. The package usage

3.1. Input

The US map on the state level is often used to compare cartogram algorithms. To generate a useful dual graph, which is a requirement of the algorithm, the input map regions have to overlap. For the `state.x77` data used in this section this can be done by correcting lines of longitude derived from the square roots of the area values (see Figure 2 left).

```
R> R.version.string

[1] "R version 3.5.1 (2018-07-02)"

R> packageVersion("recmap")

[1] '0.5.37'

R> US.map <- data.frame(x = state.center$x,y = state.center$y,
+    dx = sqrt(state.area) / 2 / (0.7 * 60 * cos(state.center$y *
+      pi / 180)), dy = sqrt(state.area) / 2 / (0.7 * 60),
+    z = sqrt(state.area), name = state.name)
R> head(US.map)

          x       y        dx       dy        z       name
1  -86.7509 32.5901  3.209890 2.704478 227.1761    Alabama
2 -127.2500 49.2500 14.005670 9.142338 767.9564     Alaska
3 -111.6250 34.2192  4.859044 4.017906 337.5041    Arizona
```

```
4  -92.2992 34.7336  3.338204 2.743370 230.4431   Arkansas
5 -119.7730 36.5341  5.902177 4.742415 398.3629 California
6 -105.5130 38.6777  4.923602 3.843727 322.8730   Colorado
```

Please note, in general, `recmap` is not transforming the geodetic datum, e.g., WGS84 or Swissgrid. Furthermore, geospatial positions have to be mapped from the earth surface to the plane. This transformation has to be done prior the cartogram generation. An overview of map projection can be found in Snyder (1997). Map projections aim to optimize towards different objectives, e.g., conformal mapping (preservation of local angles) or area mapping (preservation of area). In cartogram publications, the map projection aspect is often neglected, and conformal errors are accepted. However, studying US cartograms, a cylindrical projection seems to be the projection of choice. The R user can find support for a wide variety of adequate map projections due to using the **mapproj** package by McIlroy, Brownrigg, Minka, and Bivand (2015).

3.2. Run

The algorithm takes a `data.frame` object having the column names `c("x", "y", "dx", "dy", "z", "name")`, here the `US.map` object, as input. Additional control is given by the index order and the (dx, dy) values. The index order Π defines in which order the dual graph is explored and the (dx, dy) values define which map regions are adjacent.

```
R> library("recmap")
R> US.cartogram <- recmap(US.map)
```

3.3. Output

The `recmap` method returns an S3 object of class `c("recmap", "data.frame")` of the transformed map. Additional columns in the result contain information for topology and relative position error defined in Equations 10 and 12. The column `dfn.num` indicates in which order the dual graph has been explored.

```
R> head(US.cartogram)
```

```
           x        y       dx       dy        z       name dfs.num
1  -79.27226 10.40598 3.916894 3.300161 227.1761    Alabama      23
2 -129.22283 63.71115 8.181797 5.340748 767.9564     Alaska      49
3 -126.86923 30.51915 4.819169 3.984933 337.5041    Arizona      34
4  -87.19357 10.42302 3.994415 3.282650 230.4431   Arkansas      39
5 -137.00972 33.40013 5.311325 4.267664 398.3629 California      35
6 -115.35010 62.23315 4.851078 3.787109 322.8730   Colorado      48
  topology.error relpos.error relposnh.error
1              6    0.3893223      0.7694393
2              6    0.2883036      0.5287275
3              3    0.2074725      0.2355095
4              8    0.5599091      1.0584316
5              4    0.1761236      0.1878248
6             10    0.6831601      1.1107418
```

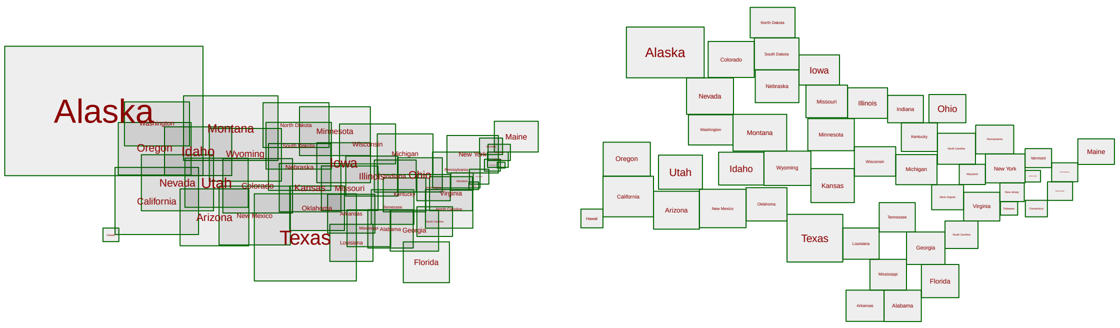

Figure 2: The "usage" example, generated from the "US State Facts and Figures" data in the **datasets** package, was used for drawing a rectangular map approximation. The input set of overlapping rectangles is shown left. A feasible solution thereof generated with `recmap` is on the right side. The state area, original size of the map region, is used as statistical value.

The output of the `recmap` function can be visualized using the S3 `plot` method for '`recmap`' objects. The output of the R code snippet can be seen in Figure 2. As default the `plot` method for '`recmap`' objects places the name attribute in the center of each rectangle. The text is scaled by using `cex = dx / strwidth(name)` as an argument for the `text` function to avoid overplotting of the labels. However small statistical values result in small label areas. This problem can be circumvented by using an interactive visualization, e.g., using **shiny** (Chang, Cheng, Allaire, Xie, and McPherson 2016) the function `hoverOpts` enables the "mouseover" feature. Please note that while the input `US.map` is not classified as an object of class '`recmap`', the `plot` method function for '`recmap`' objects cannot be used through S3 method dispatch and function `plot.recmap` has to be explicitly called.

```
R> op <- par(mfrow = c(1, 2), mar = c(0, 0, 0, 0))
R> plot.recmap(US.map, col.text = "darkred")
R> plot(US.cartogram, col.text = "darkred")
```

A summary method implements the calculation of some metadata including the objective functions.

```
R> summary(US.cartogram)


                                values
number of map regions        50.000000
area error                    0.000000
topology error              266.000000
relative position error       0.420000
screen filling [in %]        36.893585
xmin                       -146.967409
xmax                        -34.722706
ymin                          7.105818
ymax                         73.190904
```

The S3 methods, `as.recmap` and `as.SpatialPolygonsDataFrame`, enable the exchange and the manipulation of geographic data contained in the classes of the **sp** package (Pebesma and Bivand 2005; Bivand, Pebesma, and Gómez-Rubio 2013). The following code displays how a '`recmap`' object can be translated into a '`SpatialPolygonsDataFrame`' object and back.

```
R> X <- checkerboard(8)
R> all.equal(X, as.recmap(as.SpatialPolygonsDataFrame(X)))

[1] TRUE
```

4. Implementation

The RecMap algorithm is implemented in C++ using features provided by the C++-11 standard. The input and output data transfer between R and the C++ '`recmap`' class is handled by using the **Rcpp** (Eddelbuettel and François 2011) mechanism. The C++ '`recmap`' class itself consists of a `std::vector` of `map_region`. A `map_region` contains all the (x, y, dx, dy, z) values, a `std::vector` of type `int` linking to its neighbor map regions, and some additional help variables to ease the error computation. In general, the construction algorithm follows the map partition 2 (MP2) procedure described in Heilmann *et al.* (2004). The local placement function can place any rectangle next to another rectangle as demonstrated in Figure 3. The current implementation starts with the original bearing α of the two map region centers (see Figure 3 where $\beta = 0$). If the placement does not lead to a feasible solution, the angle β is added to α. β is iterating between $[0, \pi]$ with a step size of $\frac{\pi}{180}$ and a changing sign until a placement without overlap has been found. If no placement can be found, the algorithm considers all adjacent placed map regions. If also in a later step during the depth-first search (DFS) a map region cannot be placed, a non-feasible solution is accepted. This situation is often caused because the construction algorithm is hampered by the input configuration of the map regions. Solving this can be very compute-expensive and often the procedure leads to a solution which will be rejected by the metaheuristic due to the low fitness value.

Furthermore, no genetic algorithm (metaheuristic) has been implemented. Here **recmap** will use the **GA** package (Scrucca 2013) available on CRAN as demonstrated in Section 5. The most computationally expensive part is the computation of MBB intersections which has to be performed to achieve feasible solutions, multiple times, for each placement step. In the package version 0.2.1 these tests were performed by iterating over each map region. All later versions use a `std::multiset` data structure and a `std::lower_bound` algorithm of the C++ standard template library (STL) to reduce the search space.

The time complexity for one `recmap` run is $\mathcal{O}(n^2)$, where n is the number of regions. A DFS run is visiting each map region only once, and therefore it has time complexity $\mathcal{O}(n)$. For each `map_region` placement, a constant number of MBB intersection are called (max. 360). The MBB check is implemented using a `std::multiset` container, and the functions `std::insert`, `std::upper_bound`, and `std::upper_bound`. The time complexity for all of these functions is reported on `http://www.cplusplus.com/reference/stl` as $\mathcal{O}(\log(n))$. However, the worst case scenario for a range query is $\mathcal{O}(n)$, if and only if dx or dy cover the whole x or y range. The boxplots in Figure 4 (left plot) show that the number of MBB intersection test calls could be reduced by using a `std::multiset` data structure. For this benchmark, synthetic

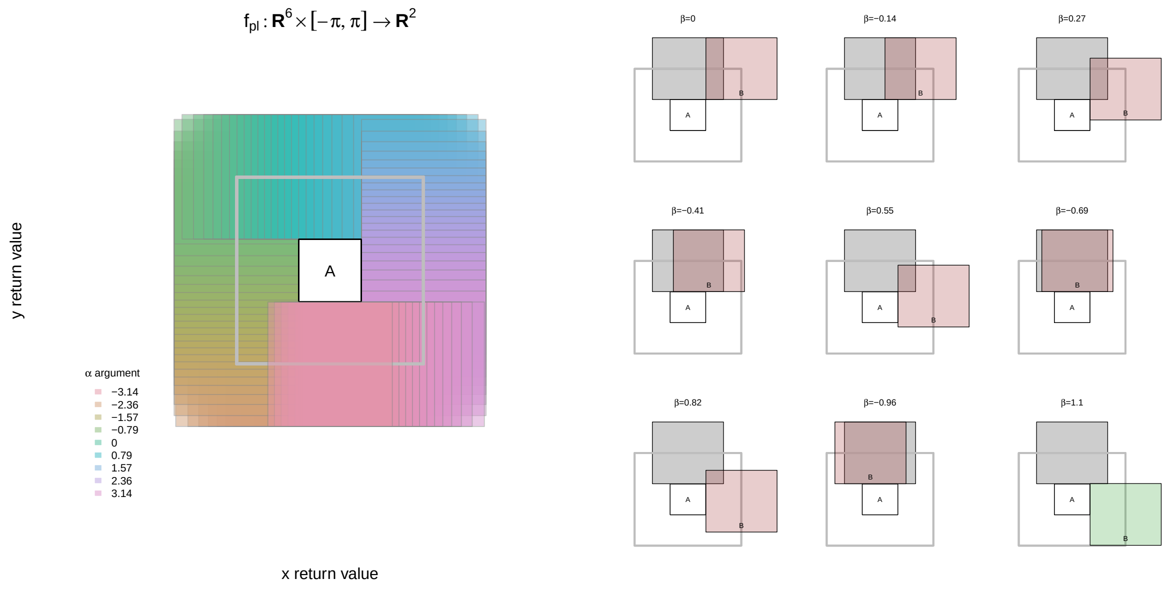

Figure 3: The rainbow colored rectangles graph the feasible positions of a local placement function f_{pl} to place a rectangle B around a given rectangle A of any angle α between $[-\pi, \pi]$ (left figure). There are special cases for quadrant I, II, III, and IV indicated by different colors in the graphic. On the right figure, B should be placed around A starting with an angle of $\frac{\pi}{4}$ (this is the initial relative position in the input map). Since the rectangle cannot be placed without overlap (indicated by red), an angle β is added.

checkerboards with map regions in the interval of $[2^2, \ldots, 20^2]$ were generated using the R function `checkerboard`. For each checkerboard size `recmap` was called 100 times using different index orderings of the checkerboard input. The index order of the input records has a direct impact how the DFS is traversing the map. This characteristic will later be used for the metaheuristic.

The benchmark was performed on an Apple MacBookPro Laptop having a 2.9 GHz Intel Core i7 CPU from 2017 using only one core. The experiments were performed on MacOSX 10.13.4, Linux Debian 9 running on a 4.9.06-amd64 kernel and a Windows 10 platform. The Linux and Windows OS instances were running as virtual machines using VirtualBox. The middle plot in Figure 4 displays the resulting mean aggregated measured data of the three (hardware/OS) systems using the two different implementations of the MBB intersection test. Besides the fact that the Linux system cannot benefit from the more efficient implementation of the MBB intersection test, the plot in the middle shows that even for an input size of $20^2 = 400$ map regions a rectangular cartogram can be computed in less than a second.

The right plot in Figure 4 was derived from the performance study (plot in the middle). It shows the number of rectangular cartograms which can be generated within one second depending on the number of map regions. The gray vertical line indicates the number of the US map regions. The ability to generate a high number of cartogram candidates in a short time period is an important requirement for any metaheuristic as is demonstrated in the next section.

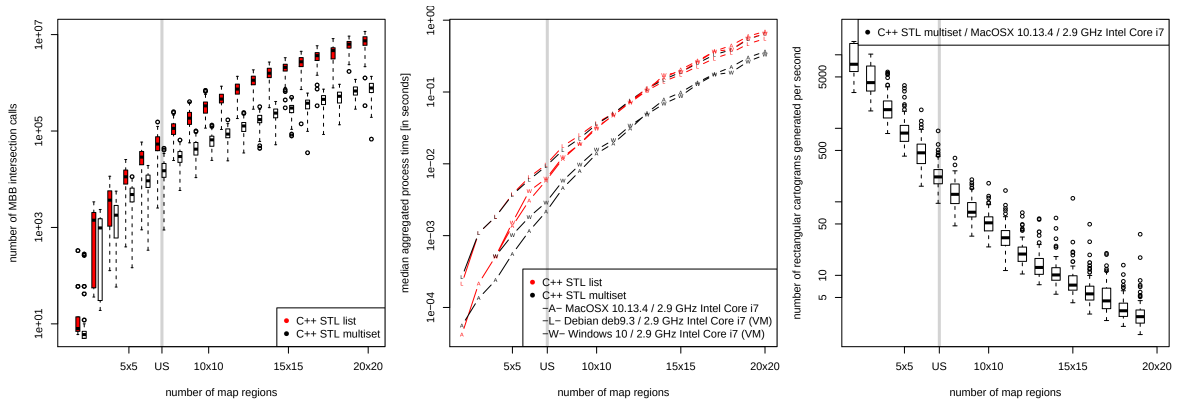

Figure 4: The boxplot (left) graphs the number of MBB intersection test calls for a given checkerboard size. The graph in the middle displays the mean aggregated measured computation time. The scatterplot (right) displays how many rectangular cartograms can be generated within one second based on the board size. All y-axes are on logarithmic scale.

5. Choose a metaheuristic

The design of the RecMap algorithm is as follows: First, compute a set of feasible solutions. In a second evaluation step, choose the best solution. In the current implementation, variations can be introduced by changing the index order Π of the input data. This order has a direct impact on the DFS traversal and leads to different cartogram layouts. The objective functions, Equations 5 to 12, can be used to evaluate the result and to define a fitness function which has to be maximized. The following R code defines the fitness function which will be used as default.

```
R> recmap:::.recmap.fitness

function (idxOrder, Map, ...)
{
    Cartogram <- recmap(Map[idxOrder, ])
    if (sum(Cartogram$topology.error == 100) > 0) {
        return(0)
    }
    1/sum(Cartogram$relpos.error)
}
<environment: namespace:recmap>
```

Other variants of fitness functions will lead to different results as shown in Heilmann *et al.* (2004, Figure 4).

Since it is not possible to compute and evaluate all permutations, which is $n!$, random experiments are conducted. In the following section, it is demonstrated how two metaheuristics, GRASP and GA, can be used to find an optimal layout for the rectangular statistical cartogram.

For a visual evaluation of the metaheuristics, proposed in this section, an 8×8 checkerboard will be used as input map. The map is generated using the `checkerboard` method.

```
R> Checkerboard <- checkerboard(8)
R> summary(Checkerboard)

                        values
number of map regions    64.00
area error                0.27
topology error              NA
relative position error     NA
screen filling [in %]   100.00
xmin                      0.50
xmax                      8.50
ymin                      0.50
ymax                      8.50
```

and can be seen in Figure 5 (left). If the assumption is made, that for each vertex, the cyclic order of edges in the contiguous cartogram remains the same as in the input map, checkerboards provide examples of sets of map regions which do not have ideal cartogram solutions (Keim *et al.* 2004, Definition 2, Lemma 1, Figure 3).

5.1. Greedy randomized adaptive search procedures

One group of optimizers that is "trivial to efficiently implement" (Feo and Resende 1995) and can directly benefit from a parallel environment is called *greedy randomized adaptive search procedures* (GRASP). The R method `recmapGRASP` defines a generic GRASP implementation as described in Feo and Resende (1995, Figure 1). The `recmapGRASP` function generates a set of rectangular cartograms which have different layouts caused by the random sampling. Each cartogram is evaluated. The best candidate is saved. The following command will generate a cartogram solution based on a GRASP metaheuristic.

```
R> set.seed(1)
R> res.GRASP <- recmapGRASP(Checkerboard)
R> plot(res.GRASP$Cartogram,
+    col = c("white", "white", "white", "black")[res.GRASP$Cartogram$z])
```

A drawing of the cartogram can be found in Figure 5 (middle).

For some types of input maps, GRASP can generate amazing results in a short time. As it can be seen in Figure 5, for the checkerboard, GRASP is outperformed by the genetic algorithm, introduced in the next section. The plot in Figure 6 (right) shows that the solution process runs too fast into saturation.

5.2. Lessons learned from biological evolution

In this paragraph, a constraint-based genetic algorithm (GA) as discussed in Heilmann *et al.* (2004) is used as metaheuristic. Here the construction heuristic benefits from the existence of the **GA** package by Scrucca (2013). The GA configuration used for `recmap` was directly derived from the *traveling salesperson problem* (TSP) example (Scrucca 2013, Section 4.8) using the *permutation* type of the `ga` method. As genotype the index order Π of the input map is used. The following command generates an almost perfect rectangular cartogram for the checkerboard map having 64 map regions on the author's laptop (MacBook Pro from 2017, 2.9 GHz Intel Core i7) within 60 seconds.

```
R> recmap.GA <- ga(type = "permutation", fitness = recmap:::.recmap.fitness,
+    Map = Checkerboard, min = 1, max = nrow(Checkerboard), popSize = 50,
+    maxiter = 300, maxFitness = 1.7, maxiter = 300, pmutation = 0.25)
```

The metaheuristic stops when a maximum number of iteration has been performed or the fitness value is higher than 1.7. Having reached a fitness value of 0.342 using the `recmap.fitness` function the result in Figure 5 (right) looks like an almost "optimal" solution.

The `recmapGA` function is a higher level wrapper function to glue the `recmap` construction heuristic with the metaheuristic `ga`.

```
R> res.GA <- recmapGA(Checkerboard, popSize = 50, run = 300, maxiter = 300,
+    seed = 3)
R> summary(res.GA$Cartogram)

                            values
number of map regions    64.0000000
area error                0.0000000
topology error          294.0000000
relative position error   0.0500000
screen filling [in %]    65.7237172
xmin                      0.4088612
xmax                      9.9656942
ymin                     -0.7320998
ymax                      9.4571887

R> plot(res.GA$Cartogram,
+    col = c("white", "white", "white", "black")[res.GA$Cartogram$z])
```

The `recmapGA` function returns a list of the input `Map`, the solution of the GA, and a '`recmap`' object containing the cartogram. The resulting cartograms using a GA can be seen in Figure 5 (right). The red line in Figure 6 (left) indicates in which order the rectangles were placed using the DFS numbering. The red • symbol marks the first placed rectangle and the ⋄ the last one.

Figure 7 illustrates the variability in solutions, dependent on the initial seed value. The experiment was repeated twice to demonstrate the effect that the same seed values lead to the same permutation order Π and finally to the same cartogram construction. As shown above, the results of the recmap implementation are reproducible on the same platform. However, due to the numerical ill-condition of the problem special care has to be taken for the computation of the fitness value. Small differences in the fitness values on different platforms cause error propagation through iterations of the metaheuristic and derive a different solution sequence Π and cartogram drawing.

The rectangular map approximations in Figure 8 demonstrate the continuous improvement of the feasible solutions with an increasing number of generations using the GA as metaheuristic for the data used in Section 3. The code below defines a weighted fitness function.

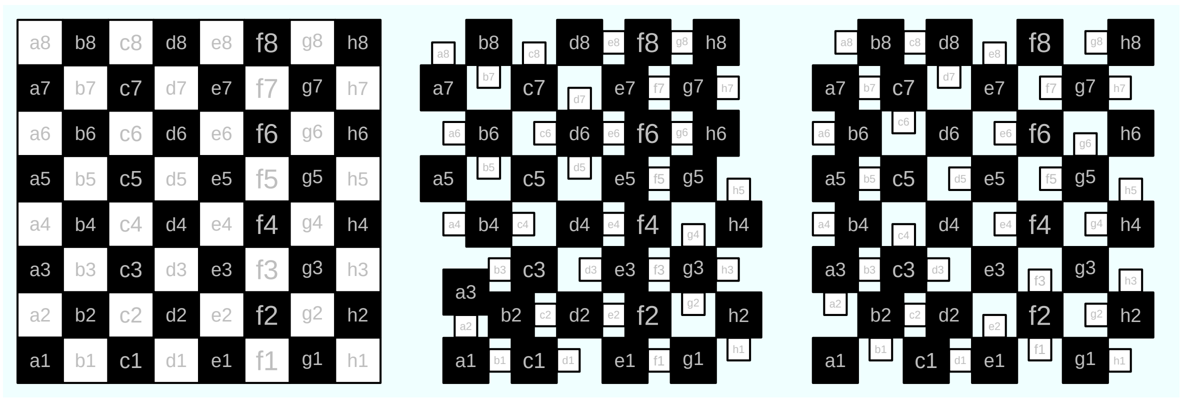

Figure 5: A comparison of the input map, GRASP, and GA is displayed. As input map, a 8 × 8 checkerboard (left) has been generated. The area of a black box needs to be four times as large as the area of a white box. The cartogram in the middle proposes a solution generated by a GRASP metaheuristic. The right cartogram graphs a solution computed by a genetic algorithm within one minute.

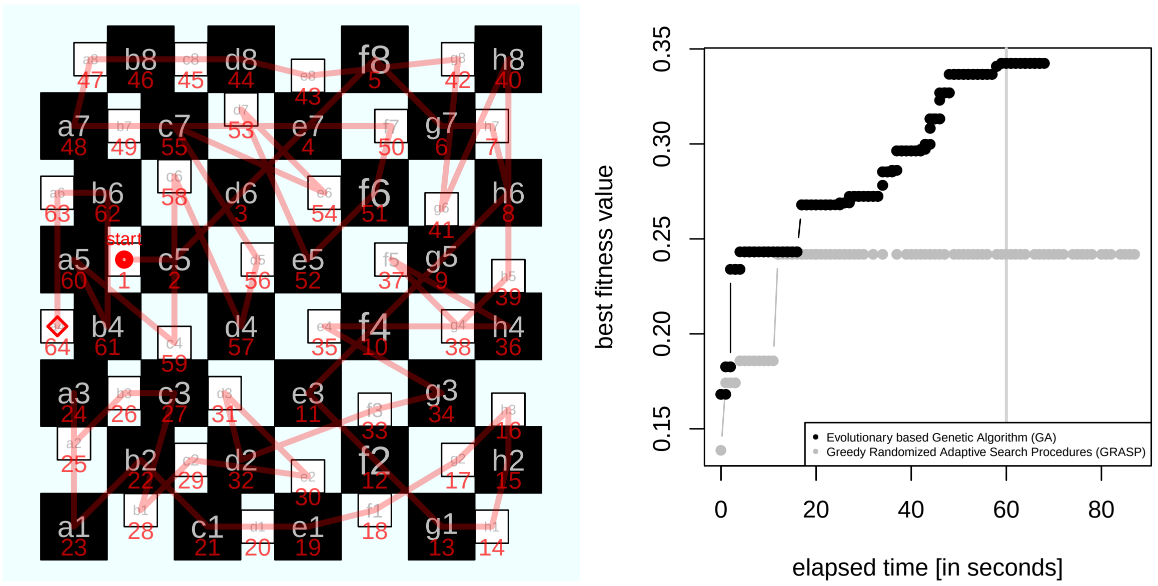

Figure 6: On the checkerboard cartogram drawn on the left, the red lines trace the order of each rectangle placement step of the construction algorithm. The right scatterplot graphs the best fitness value computed by `recmap.fitness` versus the elapsed time of the two meta-heuristics. Each • represents an iteration. The GRASP reaches the plateau after a few iterations.

```
R> fitness.weighted <- function (idxOrder, Map, ...) {
+    Cartogram <- recmap(Map[idxOrder, ])
+    if (sum(Cartogram$topology.error == 100) > 0) {
+      return(0)
+    }
+
```

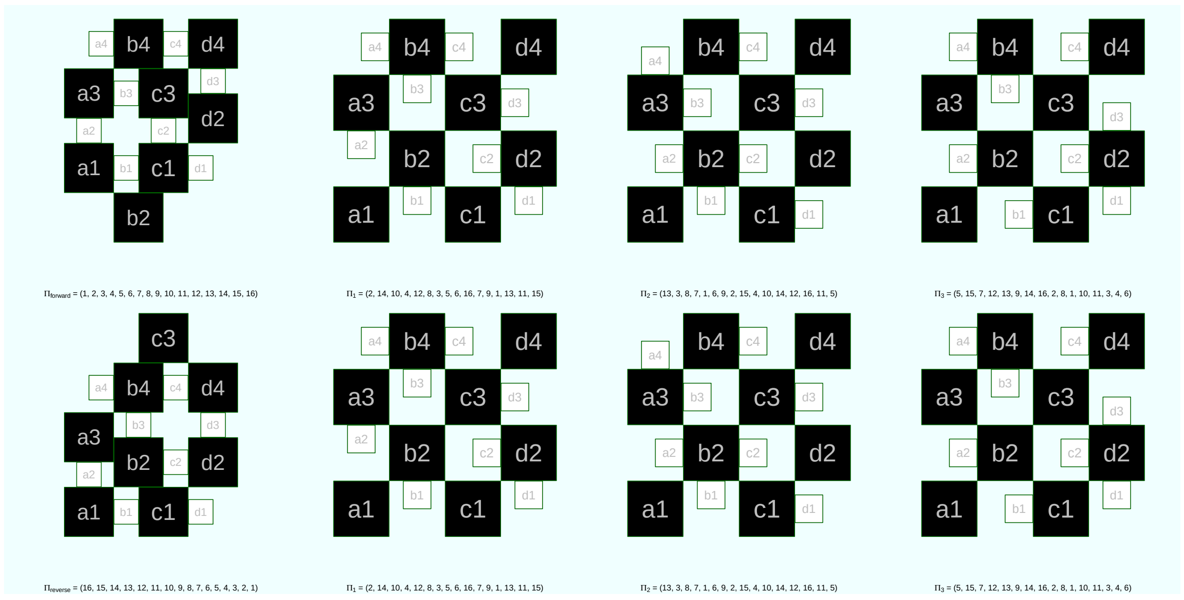
$\Pi_{forward}$ = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16)
Π_1 = (2, 14, 10, 4, 12, 8, 3, 5, 6, 16, 7, 9, 1, 13, 11, 15)
Π_2 = (13, 3, 8, 7, 1, 6, 9, 2, 15, 4, 10, 14, 12, 16, 11, 5)
Π_3 = (5, 15, 7, 12, 13, 9, 14, 16, 2, 8, 1, 10, 11, 3, 4, 6)
$\Pi_{reverse}$ = (16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1)
Π_1 = (2, 14, 10, 4, 12, 8, 3, 5, 6, 16, 7, 9, 1, 13, 11, 15)
Π_2 = (13, 3, 8, 7, 1, 6, 9, 2, 15, 4, 10, 14, 12, 16, 11, 5)
Π_3 = (5, 15, 7, 12, 13, 9, 14, 16, 2, 8, 1, 10, 11, 3, 4, 6)
a1
b1
c1
d1
a2
b2
c2
d2
a3
b3
c3
d3
a4
b4
c4
d4

Figure 7: This graphical example illustrates the variability in solutions, dependent on the initial seed value in $\Pi_{\texttt{seed}}$. The left column displays forward- and reverse index orders. All other columns show the results from duplicate seeds $\{1, 2, 3\}$ to demonstrate that the same seed will lead to the same index order and the identical layout of the cartogram.

```
+     S <- summary(Cartogram)
+     dT <- max(Cartogram$topology.error)
+     dR <- S[4, ]
+     dE <- (100 - S[5, ]) / 100
+
+     1 / (c(0.2, 0.6, 0.2) %*% c(dT, dR, dE))
+   }
R> set.seed(2)
R> US.map.best <- recmapGA(Map = US.map, fitness = fitness.weighted,
+    maxiter = 100, maxFitness = 100, popSize = 50, keepBest = TRUE,
+    pmutation = 0.35, parallel = TRUE)
```

Note that the metaheuristic of the space filling *quad tree* (Finkel and Bentley 1974) based RecMap MP1 variant could also be realized by using the **GA** package. Here, instead of a permutation, a binary representation of decision variables has to be chosen. This can be done by setting the `type` attribute of the `ga` function to `"binary"`. The genotype, given as a binary vector, is defining the split type of the *quad tree* data structure. A `1` is applying a vertical split, while a `0` triggers a horizontal split.

6. Application

This section applies package **recmap** to some real world maps having numbers of map regions of different magnitudes and different kind of topology. Beside Figures 1 and 9 the examples focus more on the demonstration of the drawing characteristics of the `recmap` method itself and

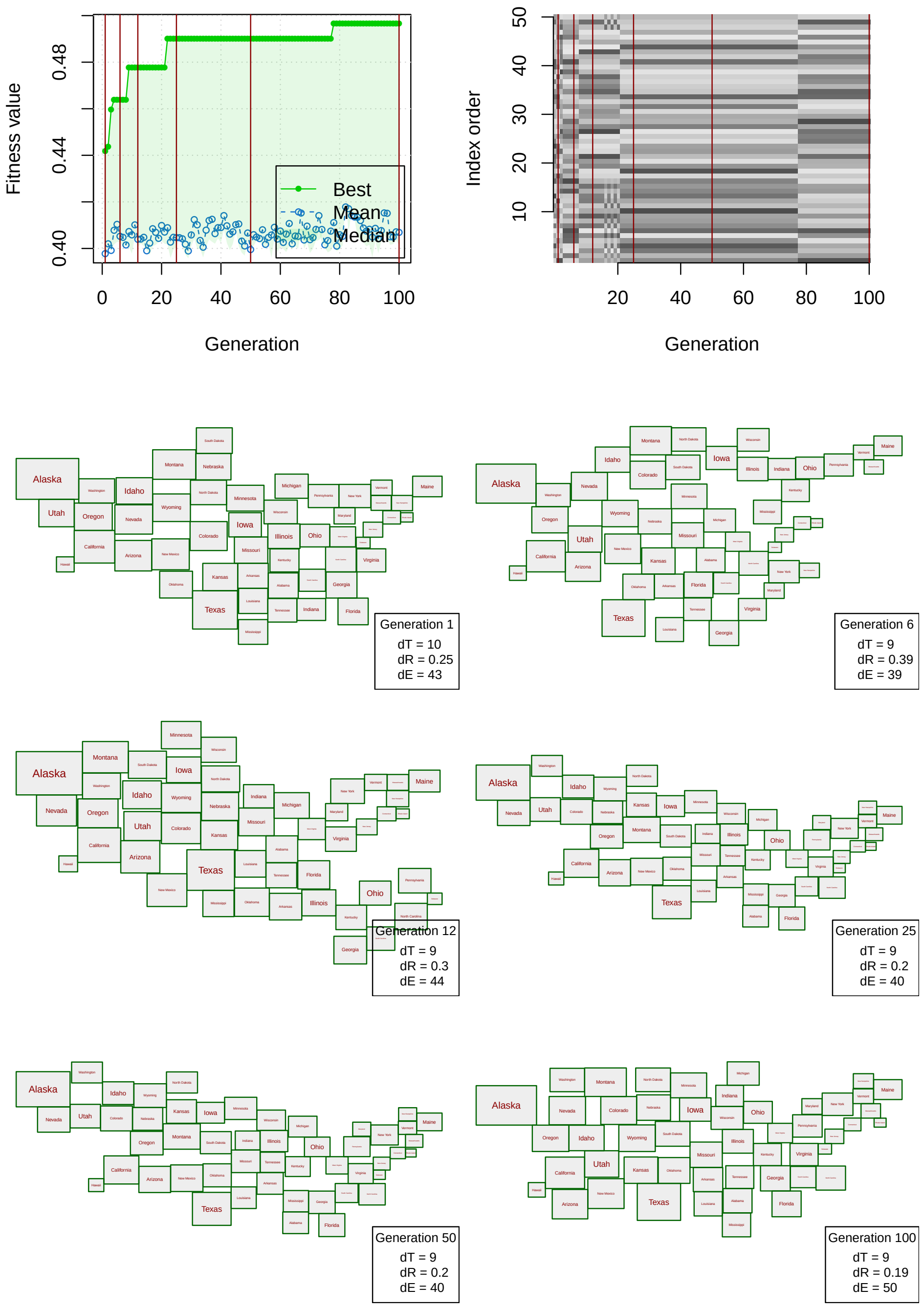

Figure 8: The plot on the top left displays the fitness value during the increasing generation of the genetic algorithm. The image plot on the top right visualizes the genotype (Π) change from one generation to the other using a gray colormap for encoding the index order. The six phenotypes visualize the improvement of the solution with an increasing number of generations.

less on the information visualization scopes. Examples of combining cartogram drawings with pixel visualization techniques can be found in Panse, Sips, Keim, and North (2006). Figure 13 demonstrates how '`recmap`' objects can be transformed into '`SpatialPolygonsDataFrame`' objects.

US state facts and figures based cartograms are displayed in Figure 9. The data are available from the data frame `state.x77`. On the cartograms, two statistical data are displayed using the area and the color of a map region. The colormap was generated by using the `heat_hcl` function of the **colorspace** package by Zeileis, Hornik, and Murrell (2009) (red is low; white is high). The code below is a wrapper function for the `recmap` and the `ga` functions. A tuple of `state.x77` column names is given as input.

```
R> recmap_state_x77 <- function(input, Map = US.map, DF = state.x77,
+    cm = heat_hcl(10)) {
+    # Join map and data.frame
+    Map <- cbind(Map, DF, match(Map$name, row.names(DF)))
+    attr(Map, "Map.name") <- "U.S."
+    attr(Map, "Map.area") <- input$area
+
+    # Filter
+    Map <- Map[!Map$name %in% c("Hawaii", "Alaska"), ]
+
+    # Set attribute for desired area
+    Map$z <- Map[, input$area]
+
+    res <- recmapGA(Map = Map, popSize = 300, maxiter = 30, run = 10)
+
+    # Set attribute for the coloring
+    S <- Map[res$GA@solution[1, ], input$color]
+    col.idx <- round((length(cm) - 1) * (S - min(S))/(max(S) - min(S))) + 1
+
+    # Have fun
+    plot(res$Cartogram, col = cm[col.idx], col.text = "black")
+    legend("bottomleft", c(paste("area:", input$area),
+      paste("color:", input$color)), cex = 1.5)
+
+    res
+  }
```

As input map, the `US.map` defined in Section 3 is used. The lines below generate the cartograms.

```
R> op <- par(mfrow = c(4, 1), mar = rep(0.25, 4), bg = "white")
R> set.seed(1)
R> recmapGA.x77 <- lapply(list(list(color = "Area", area = "Population"),
+    list(color = "HS Grad", area = "Murder"), list(color = "HS Grad",
+      area = "Income"), list(color = "Life Exp", area = "Illiteracy")),
+    recmap_state_x77)
R> par(op)
```

The fitness values versus the generations are graphed using the plot method of the '`GA`' class.

```
R> op <- par(mar = c(5, 5, 3, 3), mfrow = c(4, 1))
R> res <- lapply(cartogram.x77, function(x) {
+      plot(x$GA)
+  })
R> par(op)
```

US population cartograms on county level showing cartograms of California, Colorado, Florida, New Jersey, and New York are displayed in Figure 10. The map material was extracted from the **maps** package by Becker, Wilks, Brownrigg, Minka, and Deckmyn (2016) and the population data were retrieved from the **noncensus** package by Ramey (2014). The map regions were joined over the FIPS (Federal Information Processing Standard) county codes using the `counties` data frame. The cartograms were generated by using the genetic algorithm as metaheuristic.

An interactive **shiny** (Chang *et al.* 2016) web application is available by running the code snippet below. It provides more combinations of parameter settings, attributes, and maps drawn in Figures 5, 8, 9, and 10.

```
R> library("shiny")
R> recmap_shiny <- system.file("shiny-examples", package = "recmap")
R> shiny::runApp(recmap_shiny, display.mode = "normal")
```

The last three application examples use the `recmapGA` function and the following main parameter setting: one iteration and a population size of 64. Table 2 lists other significant operational parameters. The seed values were derived by a greedy heuristic to have a proper starting construction sequence and thereby avoid intensive computing of the replication code of the manuscript. In praxis, it turned out that a population size similar to the number of map regions and the maximum number of iteration set to no more than a couple of hundred are useful initial values.

A population cartogram of Switzerland on community (Gemeinde) level is drawn in Figure 11. The rectangles of the original map were extracted from an ESRI shape file of the map data Landschaftsmodelle: GG25 from the Federal Office of Topography (swisstopo) using **shapefiles** by Stabler (2013). The following attributes were extracted for each map region: `box`, `Gemeindecode`, and `Gemeindename`. There are 2300 rectangles to place. The statistical values (population 2013, published in 2015) were downloaded from Swiss Statistics (Regionalporträts: Kennzahlen aller Gemeinden (je-d-21.03.01) Bundesamt für Statistik BFS; `http://www.bfs.admin.ch/bfs/portal/de/index/regionen/02/daten.html`) and joined by the `Gemeindecode` attribute with the swisstopo map.

A Swiss railway passenger frequency cartogram is graphed on the bottom of Figure 12. The visualization above shows the overlapping rectangles of all 724 geo-locations which define the pseudo dual of the map. The data were retrieved from `https://data.sbb.ch/explore/` and contain already the longitude and latitude coordinates of the railway main

`Map.name`	`Map.area`	`Map.number.regions`	`Map.error.area`	`GA.population.size`	`GA.number.generation`	`GA.pmutation`	`GA.fitness`	`GA.parallel`	`GA.number.recmaps_a_second`	`Sys.compute.time`	`Sys.machine`	`Sys.sysname`
US new jersey	population	21	0.38	105	25	0.25	0.26	FALSE	562.40	4.70	x86_64	Linux
US	population	48	0.41	300	14	0.25	0.12	FALSE	224.90	18.70	x86_64	Linux
US	murder	48	0.37	300	23	0.25	0.12	FALSE	222.70	31.00	x86_64	Linux
US	income	48	0.29	300	29	0.25	0.12	FALSE	210.70	41.30	x86_64	Linux
US	illiteracy	48	0.40	300	15	0.25	0.12	FALSE	220.00	20.50	x86_64	Linux
US	area	50	0.17	50	100	0.35	0.50	FALSE	179.20	27.90	x86_64	Linux
US california	population	58	0.62	290	25	0.25	0.07	FALSE	199.50	36.30	x86_64	Linux
US new york	population	62	0.67	310	66	0.25	0.10	FALSE	143.70	142.40	x86_64	Linux
checkerboard 8 x 8	1:4	64	0.27	640	10	0.25	0.21	FALSE	60.60	105.60	x86_64	Linux
US colorado	population	64	0.68	320	44	0.25	0.06	FALSE	166.20	84.70	x86_64	Linux
US florida	population	68	0.44	340	28	0.25	0.10	FALSE	157.90	60.30	x86_64	Linux
UK	number of electorates	370	0.57	64	1	0.25	0.01	TRUE	17.10	3.70	x86_64	Linux
SBB	passagier frequency	724	0.67	64	1	0.25	0.46	TRUE	6.80	9.40	x86_64	Linux
CH	population	2300	0.59	64	1	0.25	0.17	TRUE	0.40	151.50	x86_64	Linux

Table 2: The spreadsheet provides a summary of the statistical rectangular cartograms drawn in Figures 5, 8, 9, 10, 11, 12, and 13 ordered by the number of map regions. All listed rectangular cartograms were processed on Intel Core i5-2500 CPU @ 3.30GHz having four cores running Debian 9 GNU/Linux.

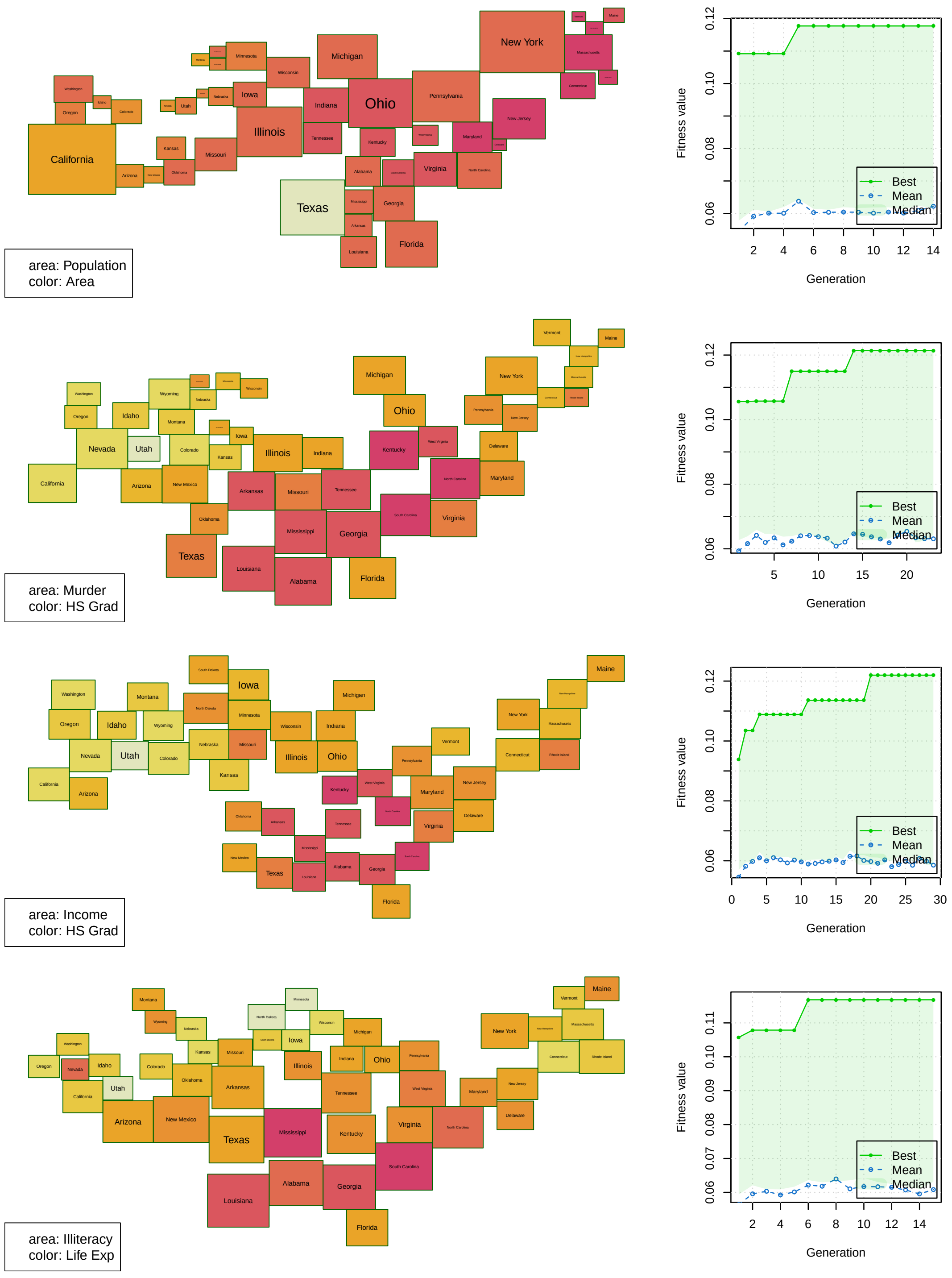

Figure 9: Rectangular statistical cartograms using the "US State Facts and Figures" dataset are drawn. The plots on the right column display the fitness values versus the generation of the genetic algorithm during the optimization process.

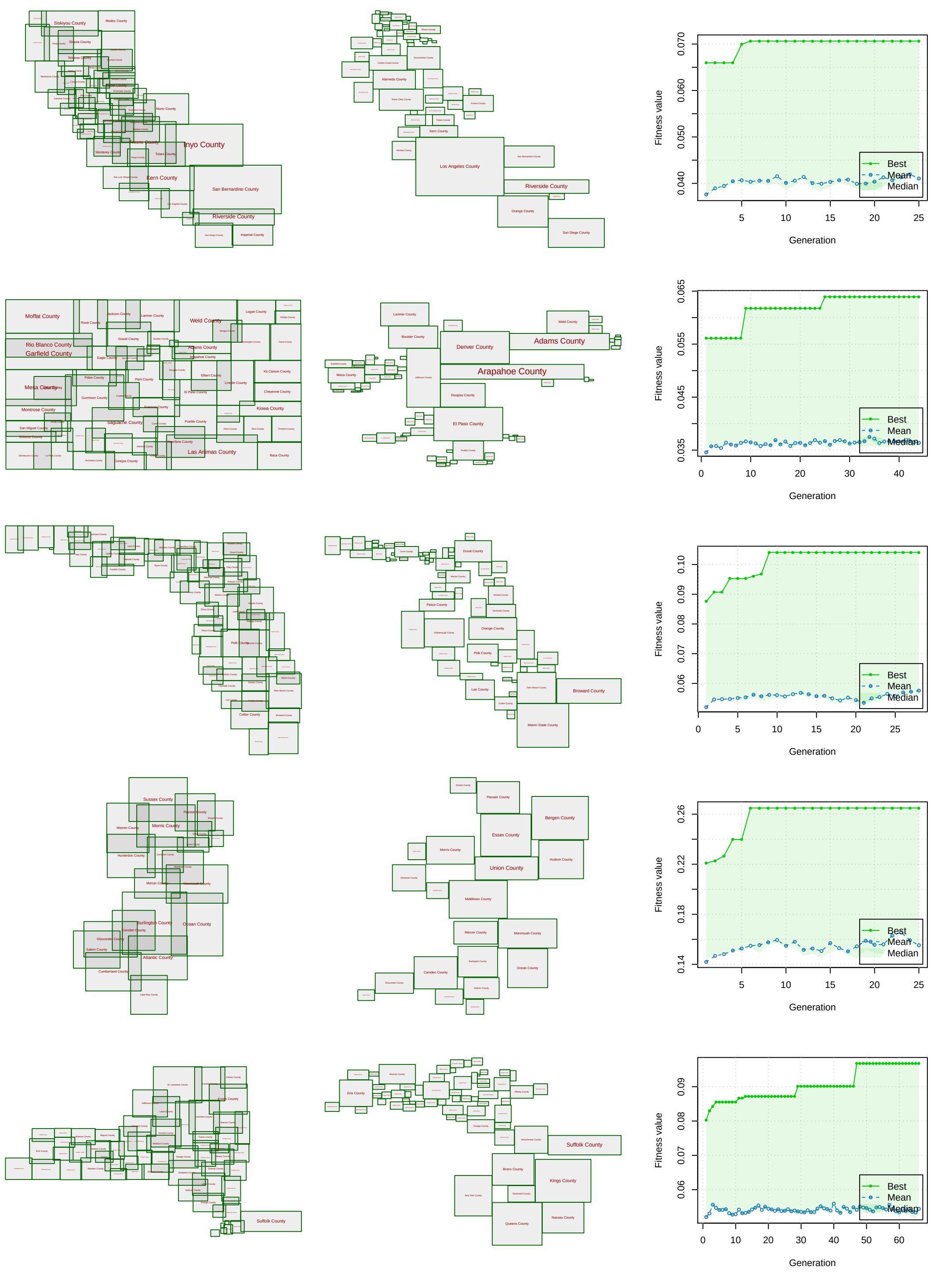

Figure 10: US input maps (left) of California, Colorado, Florida, New Jersey, and New York on county level were used as input to compute 2010 census population cartograms (middle; top to bottom). On the right column, the fitness values versus the generations are displayed.

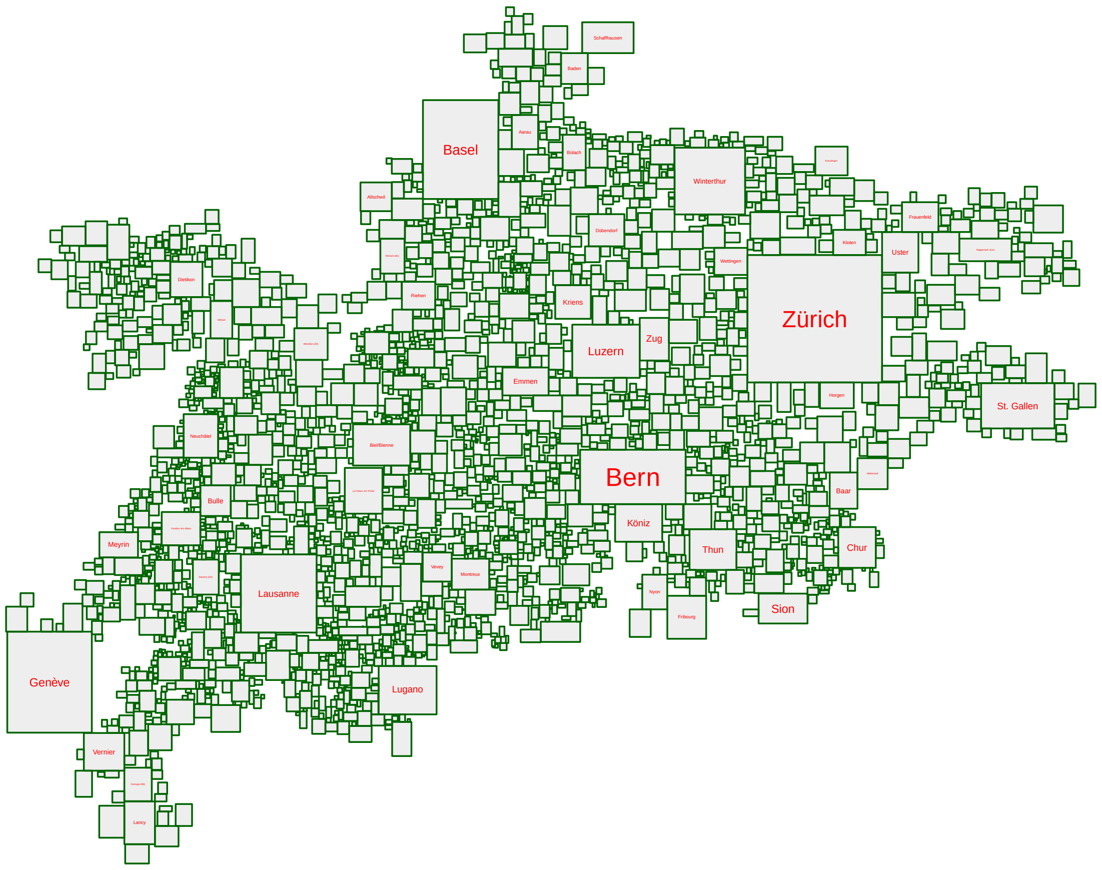

Figure 11: A rectangular population cartogram of Switzerland is shown. Map data source: Swiss Federal Office of Topography using Landscape Models/Boundaries GG25 (`http://www.toposhop.admin.ch/en/shop/products/landscape/gg25_1`, downloaded 2016-05-01); statistical data: Bundesamt für Statistik (BFS), Website Statistik Schweiz (`http://www.bfs.admin.ch/bfs/portal/de/index.html`), downloaded file `je-d-21.03.01.xls` (`http://www.bfs.admin.ch/bfs/portal/de/index/regionen/02/daten.html`) on 2016-05-26.

station and stops. The fitness function below weights the relative position error of regions with a higher traveler frequency more than map regions with a lower travel frequency.

```
R> fitness.SBB <- function(idxOrder, Map, ...) {
+    Cartogram <- recmap(Map[idxOrder, ])
+    if (sum(Cartogram$topology.error == 100) > 1){return (0)}
+    1 / sum(Cartogram$z / (sqrt(sum(Cartogram$z^2))) *
+      Cartogram$relpos.error)
+  }
```

The UK Brexit EU-referendum is shown as a final example in Figure 13. The UK boundary file was downloaded from `https://census.edina.ac.uk/` and joined by the column name `geo_code` and `Area_Code` with the outcome of the referendum downloaded through

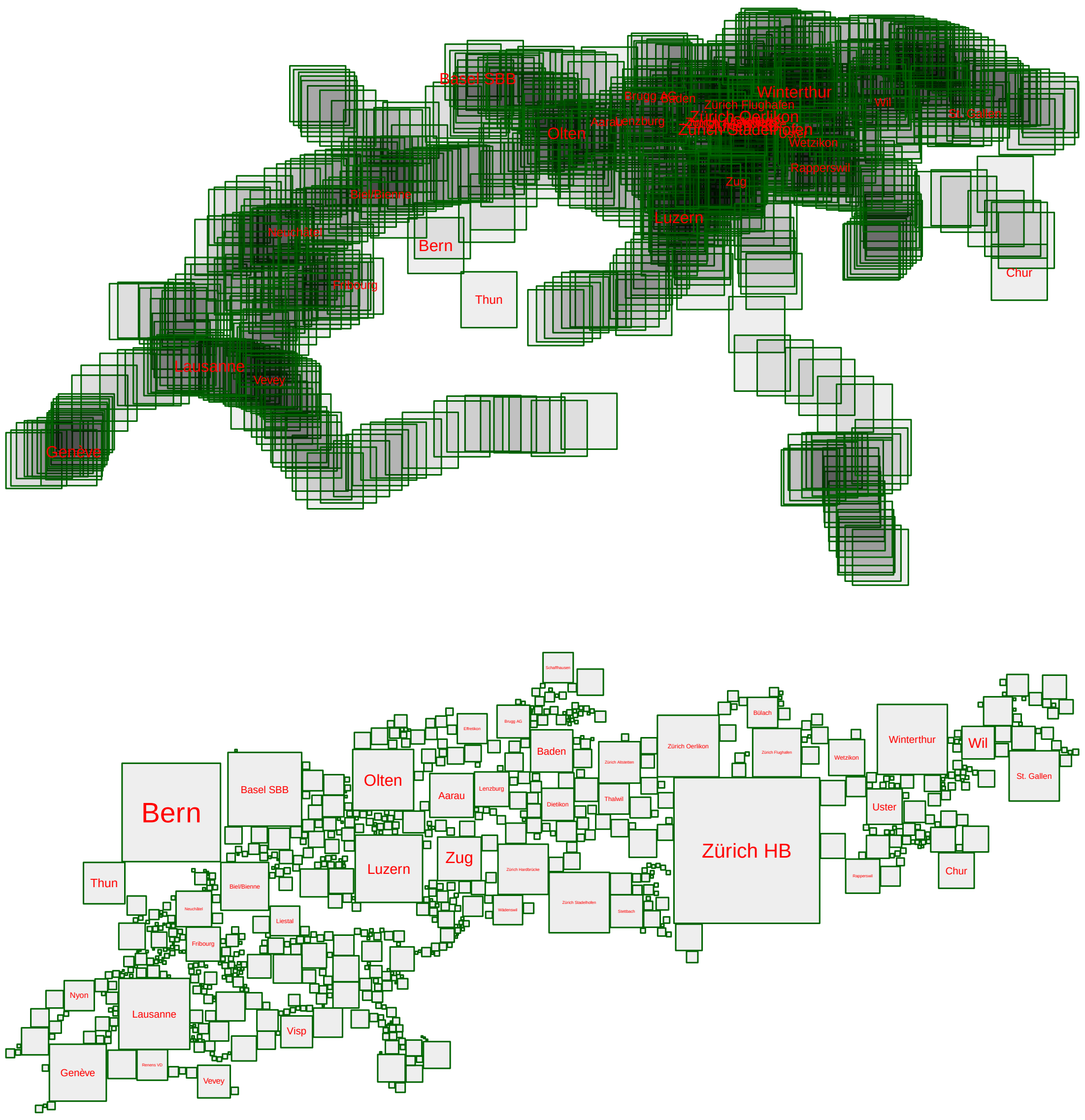

Figure 12: A Swiss railway passenger frequency cartogram is shown on the lower map. The graphic on the top displays the overlapping rectangles of the input map. Source: http://sbb.ch/, 2016-05-12.

http://www.electoralcommission.org.uk/ on July 3rd. This example also demonstrates the usage of the **sp** package by Bivand *et al.* (2013).

Through using the S3 method `as.SpatialPolygonsDataFrame` the '`recmap`' instance, `UK$Map`, has been transformed into a '`SpatialPolygonsDataFrame`' object.

```
R> DF <- data.frame(Pct_Leave = UK$Map$Pct_Leave, row.names = UK$Map$name)
R> spplot(as.SpatialPolygonsDataFrame(UK$Map, DF),
+    col.regions = diverge_hcl(16, alpha = 0.5),
+    main = "Input England/Wales/Scottland")
```

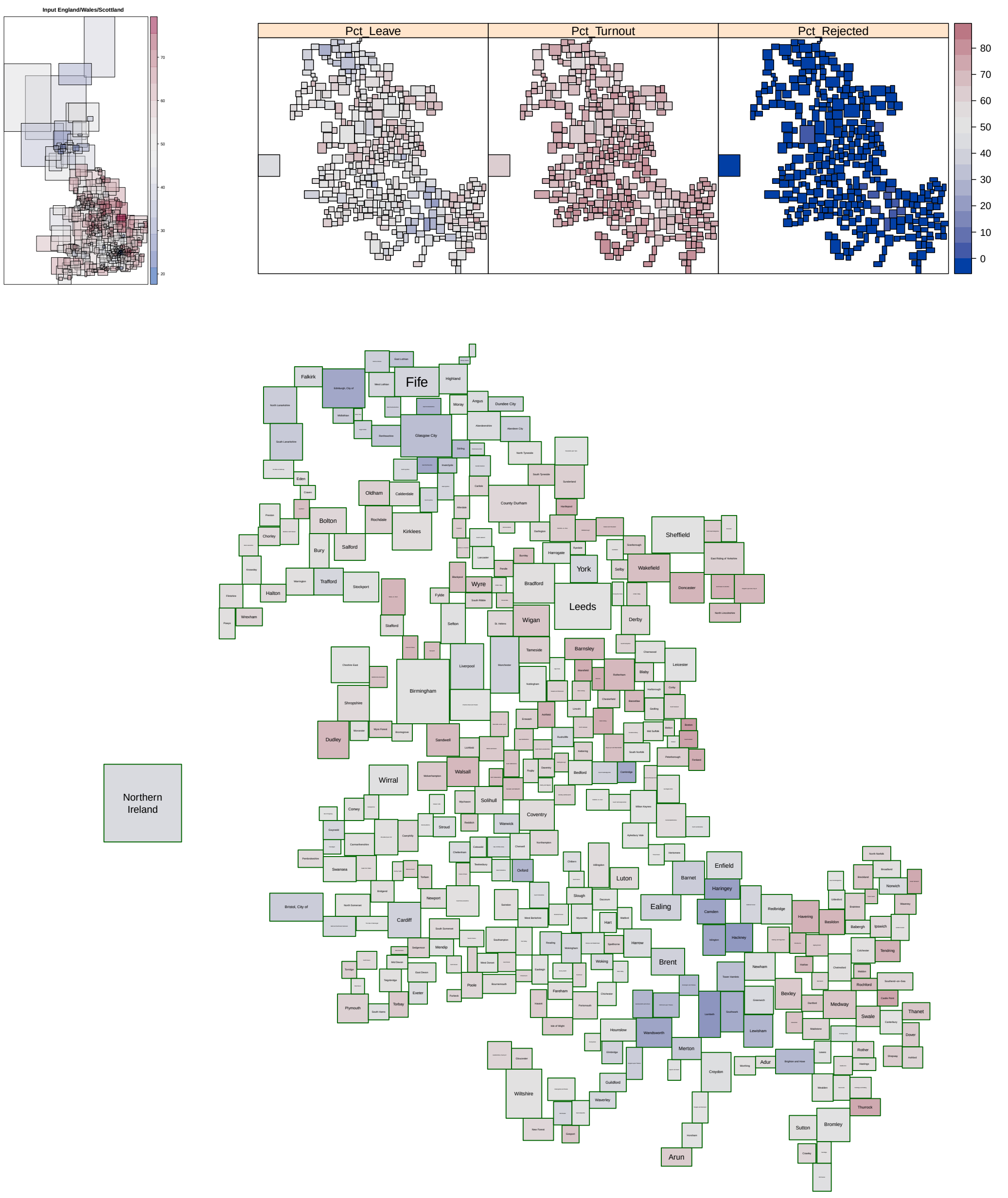

Figure 13: The outcome of the UK Brexit EU-referendum is displayed. Northern Ireland was manually added. The overlapping MBBs of the input map are displayed in the top left. On all other plots, the region areas represent the number of the electorates. The colors are indicating the outcome of the referendum (blue: remain/red: leave; the lower the color intensity the closer is the outcome to 50%:50%). Other attributes represented as percentages (Pct) are displayed using the `spplot` of the **sp** package (top right).
Copyright: Contains National Statistics data © Crown copyright and database right 2016. Contains NRS data © Crown copyright and database right 2016. Source: NISRA: Website: http://www.nisra.gov.uk/. Contains OS data © Crown copyright [and database right] (2016).

The following code snippet applies the **sp** package's `summary` and `spplot` methods after adding the NI record.

```
R> DF <- rbind(data.frame(Pct_Leave = UK$Map$Pct_Leave,
+      Pct_Turnout = UK$Map$Pct_Turnout, Pct_Rejected = UK$Map$Pct_Rejected,
+      row.names = UK$Map$name),
+    data.frame(Pct_Leave = 44.22, Pct_Turnout = 62.69, Pct_Rejected = 0.05,
+      row.names = "Northern\nIreland"))
R> UK.sp <- as.SpatialPolygonsDataFrame(add_NI(UK.recmap), DF)
R> summary(UK.sp)

Object of class SpatialPolygonsDataFrame
Coordinates:
        min     max
x -554449.5 1073563
y -596291.6 1072578
Is projected: NA
proj4string : [NA]
Data attributes:
   Pct_Leave      Pct_Turnout     Pct_Rejected
 Min.   :21.38   Min.   :56.25   Min.   :0.03000
 1st Qu.:47.38   1st Qu.:70.19   1st Qu.:0.06000
 Median :54.34   Median :74.30   Median :0.07000
 Mean   :53.29   Mean   :73.69   Mean   :0.07283
 3rd Qu.:60.49   3rd Qu.:77.89   3rd Qu.:0.08000
 Max.   :75.56   Max.   :83.57   Max.   :0.24000

R> spplot(UK.sp, col.regions = diverge_hcl(19)[1:16], layout = c(3, 1))
```

7. Summary

This article introduces the CRAN **recmap** package which implements the RecMap MP2 algorithm. This method generates rectangular statistical cartograms. Two outstanding features of the implemented algorithm are: the areas of the map regions represent the exact statistical value without any area error, and the ratios of the map regions are preserved. These constraints are important for the correct interpretation of the geography-related statistical data. It is evident that using these restrictions the map topology cannot be preserved. The implementation allows the generation of rectangular statistical cartograms having less than one hundred map regions within a few seconds with the support of a metaheuristic. The implementation enables an interactive exploratory data analysis. All necessary steps can be done on the R command line or by using web applications on a modern computer. It has been demonstrated, how the drawing of the cartogram can be optimized according to a fitness function by using a metaheuristic and benefiting from today's multi-core hardware and R's parallel environment. Most promising is using a fitness function which is derived from the relative position error objective function. It should also be highlighted that the method can read a spreadsheet containing the geographic location. It does not require any complex polygon mesh as input. The potential of the method is shown by using real world maps covering a map size of three orders magnitude and synthetic data (8 $\times$ 8 checkerboard). Table 2 provides an overview of some rectangular cartogram specification drawn in this manuscript. The **recmap** package is a powerful tool in the hand of data analysts, cartographers, or statisticians using R who want to draw their own statistical rectangular cartograms.

References

Becker RA, Wilks AR, Brownrigg R, Minka TP, Deckmyn A (2016). ***maps**: Draw Geographical Maps.* R package version 3.1.0, URL https://CRAN.R-project.org/package=maps.

Bivand RS, Pebesma E, Gómez-Rubio V (2013). *Applied Spatial Data Analysis with R.* 2nd edition. Springer-Verlag. doi:10.1007/978-1-4614-7618-4.

Brunsdon C, Charlton M (2014). ***getcartr**: Front End for **Rcartogram** Package.* R package version 1.01, URL https://github.com/chrisbrunsdon/getcartr.

Buchin K, Eppstein D, Löffler M, Nöllenburg M, Silveira R (2016). "Adjacency-Preserving Spatial Treemaps." *Computational Geometry*, **7**(1), 100–122. doi:10.20382/jocg.v7i1a6.

Buchin K, Speckmann B, Verdonschot S (2012). "Evolution Strategies for Optimizing Rectangular Cartograms." In N Xiao, M Kwan, MF Goodchild, S Shekhar (eds.), *Geographic Information Science. GIScience 2012*, volume 7478 of *Lecture Notes in Computer Science*, pp. 29–42. doi:10.1007/978-3-642-33024-7_3.

Chang W, Cheng J, Allaire J, Xie Y, McPherson J (2016). ***shiny**: Web Application Framework for R.* R package version 0.13.2, URL https://CRAN.R-project.org/package=shiny.

Dorling D (1996). *Area Cartograms: Their Use and Creation.* 1st edition. Department of Geography, University of Bristol, England.

Eddelbuettel D, François R (2011). "**Rcpp**: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. doi:10.18637/jss.v040.i08.

Feo TA, Resende MGC (1995). "Greedy Randomized Adaptive Search Procedures." *Journal of Global Optimization*, **6**(2), 109–133. doi:10.1007/bf01096763.

Finkel RA, Bentley JL (1974). "Quad Trees: A Data Structure for Retrieval on Composite Keys." *Acta Informatica*, **4**(1), 1–9. doi:10.1007/bf00288933.

Gastner MT, Newman ME (2004). "Diffusion-Based Method for Producing Density-Equalizing Maps." *Proceedings of the National Academy of Sciences of the United States of America*, **101**(20), 7499–7504. doi:10.1073/pnas.0400280101.

Heilmann R, Keim DA, Panse C, Sips M (2004). "RecMap: Rectangular Map Approximations." In *IEEE Symposium on Information Visualization*, pp. 33–40. doi:10.1109/infvis.2004.57.

Keim DA, North SC, Panse C (2004). "CartoDraw: A Fast Algorithm for Generating Contiguous Cartograms." *IEEE Transactions on Visualization and Computer Graphics*, **10**(1), 95–110. doi:10.1109/tvcg.2004.1260761.

McIlroy D, Brownrigg R, Minka TP, Bivand R (2015). ***mapproj**: Map Projections.* R package version 1.2-4, URL `https://CRAN.R-project.org/package=mapproj`.

Nusrat S, Kobourov S (2016). "The State of the Art in Cartograms." *Computer Graphics Forum*, **35**(3), 619–642. `doi:10.1111/cgf.12932`.

Panse C (2018). ***recmap**: Compute the Rectangular Statistical Cartogram.* R package version 1.0.0, URL `https://CRAN.R-project.org/package=recmap`.

Panse C, Sips M, Keim DA, North SC (2006). "Visualization of Geo-spatial Point Sets via Global Shape Transformation and Local Pixel Placement." *IEEE Transactions on Visualization and Computer Graphics*, **12**(5), 749–756. `doi:10.1109/TVCG.2006.198`.

Pebesma EJ, Bivand RS (2005). "Classes and Methods for Spatial Data in R." *R News*, **5**(2), 9–13. URL `https://CRAN.R-project.org/doc/Rnews/`.

Raisz E (1934). "The Rectangular Statistical Cartogram." *Geographical Review*, **24**(2), 292–296. `doi:10.2307/208794`.

Ramey JA (2014). ***noncensus**: U.S. Census Regional and Demographic Data.* R package version 0.1, URL `https://CRAN.R-project.org/package=noncensus`.

R Core Team (2018). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. URL `https://www.R-project.org/`.

Scrucca L (2013). "**GA**: A Package for Genetic Algorithms in R." *Journal of Statistical Software*, **53**(4), 1–37. `doi:10.18637/jss.v053.i04`.

Snyder JP (1997). *Flattening the Earth: Two Thousand Years of Map Projections.* University of Chicago Press.

Stabler B (2013). ***shapefiles**: Read and Write ESRI Shapefiles.* R package version 0.7, URL `https://CRAN.R-project.org/package=shapefiles`.

Temple Lang D (2016). ***Rcartogram**: Interface to Mark Newman's Cartogram Software.* R package version 0.2-2, URL `https://github.com/omegahat/Rcartogram`.

Tobler W (2004). "Thirty Five Years of Computer Cartograms." *The Annals of the Association of American Geographers*, **94**(1), 58–73. `doi:10.1111/j.1467-8306.2004.09401004.x`.

Van Kreveld MJ, Speckmann B (2004). "On Rectangular Cartograms." In S Albers, T Radzik (eds.), *Algorithms – ESA 2004*, volume 3221 of *Lecture Notes in Computer Science*, pp. 724–735. `doi:10.1007/978-3-540-30140-0_64`.

Van Kreveld MJ, Speckmann B (2007). "On Rectangular Cartograms." *Computational Geometry*, **37**(3), 175–187. `doi:10.1016/j.comgeo.2006.06.002`.

Zeileis A, Hornik K, Murrell P (2009). "Escaping RGBland: Selecting Colors for Statistical Graphics." *Computational Statistics & Data Analysis*, **53**(9), 3259–3270. `doi:10.1016/j.csda.2008.11.033`.

Affiliation:

Christian Panse
Functional Genomics Center Zurich UZH|ETHZ
Winterthurerstr. 190
CH-8057, Zürich, Switzerland
Telephone: +41/44/63-53912
E-mail: cp@fgcz.ethz.ch
URL: http://www.fgcz.ch/the-center/people/panse.html

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
August 2018, Volume 86, Code Snippet 1 *Submitted:* 2016-06-01
doi:10.18637/jss.v086.c01 *Accepted:* 2017-06-12