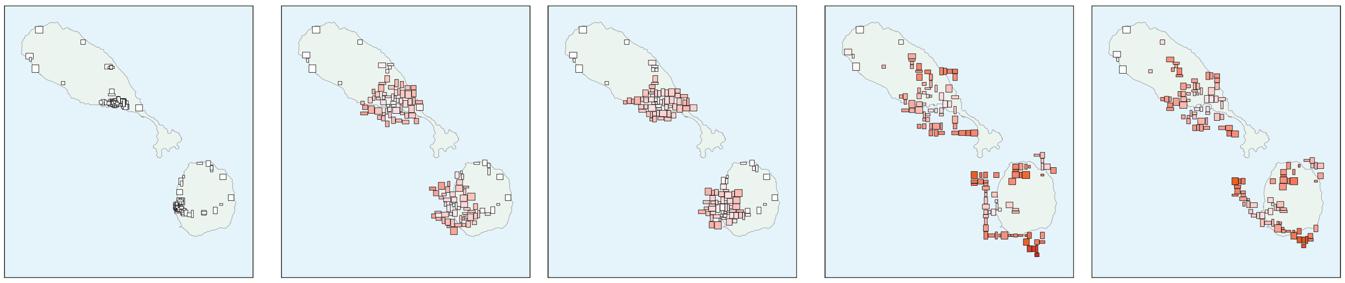
Minimum-Displacement Overlap Removal for Geo-referenced Data Visualization

M. van Garderen,[1] B. Pampel[1], A. Nocaj[1], and U. Brandes[1]

[1]University of Konstanz, Germany



(a) Actual positions

(b) Layout after overlap removal with PRISM (left) and REARRANGE (right)

(c) Layout after order-preserving overlap removal with PRISM+OR (left) and REARRANGE+OR (right)

Figure 1: *Layout results after overlap removal with two methods without (b) and two methods with (c) order preservation. Rectangles are colored by displacement with respect to their actual position (a). Both with and without order preservation* REARRANGE *results in a layout with smaller displacement and better shape preservation than PRISM.*

Abstract

Given a set of rectangles embedded in the plane, we consider the problem of adjusting the layout to remove all overlap while preserving the orthogonal order of the rectangles. The objective is to minimize the displacement of the rectangles. We call this problem MINIMUM-DISPLACEMENT OVERLAP REMOVAL (MDOR). *Our interest in this problem is motivated by the application of displaying metadata of archaeological sites. Because most existing overlap removal algorithms are not designed to minimize displacement while preserving orthogonal order, we present and compare several approaches which are tailored to our particular usecase. We introduce a new overlap removal heuristic which we call* REARRANGE. *Although conceptually simple, it is very effective in removing the overlap while keeping the displacement small. Furthermore, we propose an additional procedure to repair the orthogonal order after every iteration, with which we extend both our new heuristic and PRISM, a widely used overlap removal algorithm. We compare the performance of both approaches with and without this order repair method. The experimental results indicate that* REARRANGE *is very effective for heterogeneous input data where the overlap is concentrated in few dense regions.*

1. Introduction

In this paper we study the problem of removing overlap between labels on a map while keeping the labels close to their actual positions and in the same order. Our interest in this problem is motivated by the archaeological application of displaying information about excavation or heritage sites on a map. The same principles, however, can be applied to many other types of geo-referenced data.

1.1. Motivation

Archaeological data are often site data: a list of sites or features at various geographical coordinates and some metadata about each site. The most popular way of representing data of this kind is to use a *symbol map* [CL06], where each site is represented by a symbol that conveys (a selection of) the metadata about the site. These symbols are placed on the map at the geographical coordinates of the corresponding sites. In *proportional symbol maps* [CL06], the size of the symbols is also used to encode information about the sites. Symbol maps may represent the geological distribution of lithics sources in the research area [Mor15], site assemblages or occupation histories [GF15, Bri11], or the spatial distribution of features within a site [HM15], to name but a few examples. The problem with this approach is always the same: if sites or features are close to each other, the symbols will overlap. To make all symbols visible, they have to be scaled down and/or moved farther apart. Many GIS packages commonly used in archaeology do offer automated map production, but when it comes to the arrangement and scaling of objects they generally perform poorly [CL06].

The problem can be formulated as an instance of node overlap removal. Each archaeological site is represented by a node that has the geographical coordinates of the site as initial position. The node sizes correspond to the sizes of the metadata symbols. The objective is to find new positions for the nodes such that no two symbols overlap. Because the spatial relations and cardinal orientation among sites are important aids in the interpretation of archaeological data, we would like to keep these properties intact.

Misue et al. [MELS95] introduced three models for maintaining the mental map after layout adjustment, based on orthogonal order, proximity relations, and topology. Orthogonal order (the left-to-right and top-to-bottom order of the nodes) is particularly important to us. Archaeological research is often concerned with migration and trade routes. Changing the orthogonal order of sites changes the order in which they could be encountered on a particular route. In the context of a project on Caribbean archaeology, we are especially interested in visualizing sites in island settings. Since orthogonal order corresponds to cardinal orientation, changing the order of sites might even influence whether one place could be reached from another at all, for example if sea travel under the influence of wind and currents is involved. We therefore include orthogonal order preservation as a constraint.

If we take into account only orthogonal order, however, one important property of the layout is not yet maintained: characteristic whitespace. If the study area contains uninhabitable areas (such as the ocean between islands, or a volcano in the middle of an island), large empty spaces between groups of sites are meaningful, and we do not want to ignore them and spread the sites out evenly. We therefore aim to keep each vertex as close as possible to the actual site coordinates, which translates to minimizing the total displacement of the nodes. Maintaining the shape of the layout also makes it easier to project a map in the background in such a way that one can recognize which site belongs in which area. This again aids the archaeologist in their interpretation.

Combining the objectives and constraints described here results in the problem of removing node overlap with minimum displacement under orthogonal-ordering constraints, which we call MINIMUM-DISPLACEMENT OVERLAP REMOVAL (MDOR).

1.2. Formal definition of MDOR

Given a set of n rectangles embedded in the Euclidian plane, we consider the problem of modifying the layout to avoid intersections of the rectangles. The objective is to minimize the total displacement, under the additional constraint that the orthogonal order of the rectangles must be preserved. Let (x_i, y_i) be the initial coordinates of the center of rectangle r_i, and let (x'_i, y'_i) be the coordinates of the center after the overlap has been removed. We define the *total displacement* in the new layout as the sum of the Euclidian distances between the initial and the final position of the centers of all rectangles:

$$\text{total displacement} = \sum_{i=1}^{n} \sqrt{(x_i - x'_i)^2 + (y_i - y'_i)^2}$$

A layout adjustment is *orthogonal-order preserving* if the order of the rectangles with respect to the x- and the y-axis does not change. More formally, the order is preserved if and only if for any pair of rectangles r_i and r_j it holds that $x_i \le x_j \Rightarrow x'_i \le x'_j$ and that $y_i \le y_j \Rightarrow y'_i \le y'_j$.

1.3. Contribution

In this paper we present a new heuristic algorithm for the MDOR problem. This algorithm, which we call REARRANGE, treats the pairs of overlapping rectangles one by one. For each pair, it resolves the overlap with the smallest displacement possible while strictly keeping their orthogonal order. Since moving the rectangles of one pair might still introduce order violations with other rectangles, we further introduce a method to detect and resolve order violations in a modified layout. We extend both the popular overlap removal algorithm PRISM and our heuristic with this order repair method. In multiple experiments we then evaluate the performance of both the original and the extended algorithms.

2. Related work

The overlap removal problem has been widely studied in a variety of flavours for different applications. Two main variants of the problem can be distinguished: minimum-area and minimum-displacement overlap removal. In minimum-area overlap removal (sometimes referred to as minimum-area layout adjustment), the objective is to minimize the total drawing area needed for the new layout. This can result in very compact drawings, but is not suitable for our application on geo-referenced data where empty areas can also be meaningful. We therefore focus on the second variant on the problem, minimum-displacement overlap removal.

Minimum-area overlap removal was shown to be NP-hard [HIMF98], minimum-displacement overlap removal unsurprisingly is NP-hard as well. As a result of this, a vast amount of heuristic overlap-removal algorithms have been developed. In the following we review the main approaches for both variants of the problem, since some algorithms that were designed for the minimum-area version of the problem contain interesting ideas that could be used for any kind of overlap removal.

Along with the concept of *preserving the mental map* of a layout, Misue et al. [MELS95] presented the *Force Scan (FS)* algorithm to remove the overlap of rectangular nodes by computing forces pushing overlapping labels away from each other while keeping the total area of the drawing small. Variants that also use pulling forces to reduce white space are resulting in very compact drawings, but are not suitable for our application. Hayashi et al. [HIMF98] presented a variant called *FS'* which computes layouts with even smaller area, obtained by solving the case of multiple overlaps more space-efficiently. Both FS and FS' resolve the overlap between a pair of rectangles along a line through their centers, resulting in an unnecessarily large local displacement. The displacement is then propagated to all nodes above and to the right of the overlapping pair, which might introduce additional unnecessary displacement.

The *Force Transfer (FT)* algorithm [HL03] improves on both of these issues. Overlap is removed by horizontal or vertical movement to obtain local minimum displacement, and the displacement is propagated only to neighbors in what the authors call a *cluster sub-graph* of rectangles connected by a chain of overlaps. However, since the horizontal and vertical direction are processed one after the other, some nodes in the cluster sub-graph still are moved unnecessarily. Many other improvements to the original FS algorithm have been proposed, a survey of those and other spring methods to remove overlap can be found in [LEN05].

The *Voronoi Cluster Buster* algorithm by Lyons [Lyo92] removes overlap by iteratively moving each node toward the center of its Voronoi cell. This approach values an even distribution of the nodes more than layout similarity, making it unsuitable for our application. The *Wordle* layout algorithm by Viegas et al. [VWF09] searches a new position for an overlapping node along an outward spiral from the original position. Some improvements to this heuristic have been proposed [KLKS10, SSS*12], but none of them perform well regarding orthogonal order preservation.

Some cartogram algorithms also include removing overlap between objects. The widely used algorithm by Dorling [Dor96] defines and then moves circles in the size of values they represent. Repulsive forces push overlapping circles away from each other and attraction forces tie the circles to their original position. The iterative approach removes overlap and keeps the displacement small, but it does not preserve the orthogonal order of the circle centers.

Another branch of overlap removal algorithms is based on constrained optimization [HM97, MSTH03]. In these algorithms, separation constraints are defined between the rectangles and quadratic programming is used to solve a constrained optimization problem. Orthogonal order constraints can be added to preserve the structure, but since this method does not take proximity relations into account the shape of the layout might be changed significantly. Dwyer et al. [DMS06] apply this idea to solve the 1-dimensional version of the problem, which they call *Variable Placement with Separation Constraints (VPSC)*. To achieve better runtime for the 2-dimensional problem, they simply solve it in one dimension after the other. This results in a strong distortion into the direction that was addressed first, making this method unsuitable for our application. Hirono et al. have demonstrated that constrained optimization can also be used to remove occlusion from 3D-maps [HWAT13].

The PRISM algorithm by Gansner and Hu [GH09] minimizes *proximity stress*. It is the most commonly used overlap removal algorithm and also seems most promising for our application. Unlike most of the alternatives, it aims to maintain the shape of the pointset while removing the overlap. However, it does not maintain the orthogonal order and loses much of the relative positioning when spreading out dense areas. The approach is based on a *proximity graph*: a graph in which nodes that are close to each other share an edge. For each edge in the proximity graph the algorithm calculates a factor f_s by which it should be stretched to remove possible overlap. This stretch factor is then used, along with measures for layout similarity and compactness of the drawing, in a stress-function that should be minimized when the layout is recalculated. The computation of f_s takes into account whether the overlap is smaller in x- or y-direction and what factor would remove the overlap in this direction. However, the movement takes place by stretching the proximity edges. Multiple edges might be influencing the same node, and the other components in the stress-function also influence the movement. As a result of this, f_s has to be adapted to resolve overlap in the actual rather than the optimal movement direction. This can lead to an unnecessarily large displacement, similar to the *Force Scan* variants mentioned above. Since the new layout is computed based on stretching the edges in the proximity graph only, order violations might occur, either among nodes that do not share an edge or along edges with close to horizontal or vertical orientations.

None of the algorithms described above exactly match our objectives, so we introduce a new heuristic that is tailored to our version of the problem.

3. Heuristic ReArrange

Our heuristic algorithm ReArrange takes as input a set $R = \{r_1, \ldots, r_n\}$ of n rectangles, where rectangle $r_i = (x_i, y_i, w_i, h_i)$ is centered at position (x_i, y_i) and has size $w_i \times h_i$. The algorithm changes the coordinates of the rectangles such that, in the new layout, no two rectangles intersect. We aim to keep the total displacement of the rectangles and the number of orthogonal order violations small.

The orthogonal order of the input for both the x- and the y-dimension is determined in an initialization step in the main algorithm (lines 1-3). The order can be weak: if multiple rectangles are on the same x- or y-coordinate, they get the same rank in the order for this dimension.

Algorithm 1: ReArrange

```
  input : Set $R = \{r_1, \ldots, r_n\}$ of rectangles, where
          $r_i = (x_i, y_i, w_i, h_i)$ is centered at coordinates $(x_i, y_i)$
          and has size $w_i \times h_i$, overlap threshold value $t_0$
  output: The same set of rectangles with adjusted coordinates

1 forall $r_i \in R$ do
2     $x$-rank($r_i$) $\leftarrow$ rank of $r_i$ in weak order of $x$-coordinates;
3     $y$-rank($r_i$) $\leftarrow$ rank of $r_i$ in weak order of $y$-coordinates;

4 $P \leftarrow$ DetectOverlap($R$);

5 while $P \neq \emptyset$ do
6     Shuffle $P$;
7     foreach pair $\{r_i, r_j\} \in P$ do
8         RemoveOverlap($r_i, r_j$);

9     $P \leftarrow$ DetectOverlap($R$);
```

We use a *line-sweep* [BW80] approach to find overlapping rectangles (line 4 and 9). For each overlapping pair, we remove the overlap with local minimum displacement while taking the orthogonal order into account (lines 6-8).

The overlap removal method is based on the overlap size of the given pair of rectangles. Let the *x-overlap* of a pair of rectangles $\{r_i, r_j\}$ be the minimum displacement needed to remove their overlap in the x-dimension while taking the orthogonal order into account. Figure 2 shows two overlapping rectangles r_i and r_j. If they are already in the correct order with respect to the x-dimension, the x-overlap is simply the distance between the right border of r_i and the left border of r_j, as illustrated in Figure 2a. If the rectangles are currently reversed with respect to their original order, the x-overlap is the distance between the left border of r_i and the right border of r_j, as illustrated in Figure 2b. If two rectangles have the same rank in the x-dimension we cannot remove the overlap in this dimension without violating the order. In this case the x-overlap is set to infinity. The *y-overlap* is defined analogously.

We define the *overlap size* of a pair of rectangles as the minimum of their x-overlap and y-overlap. For rectangles that do not overlap, the overlap size is 0. If the overlap size is greater than 0 but less than the overlap threshold t_0, we set the size to t_0. This is to prevent an infinite loop in which the movements keep getting smaller with every iteration but never reach 0. Note that for rectangles that have the same rank in both dimensions, which means they are at the same initial position, the overlap size is positive infinity. Moving such rectangles apart always violates the order in at least one dimension, so this is not allowed to occur in the input.

The REMOVEOVERLAP routine takes a pair of rectangles $\{r_i, r_j\}$ as input and changes their coordinates to remove the overlap between them. It consists of two steps:

1. Compute the overlap size as described above, and the corresponding dimension (x or y).
2. If the overlap size is greater than 0 (meaning there is an overlap) and less than infinity (meaning the overlap can be removed), move the lower ranked rectangle to the left or down (depending on the dimension) by half the overlap size, and move the higher ranked rectangle to the right or up by half the overlap size.

The thick grey rectangles in Figure 2 show the new positions after overlap removal.

Moving two rectangles apart to resolve their overlap might introduce new overlap with other rectangles, so we iteratively repeat the overlap detection and removal steps until there are no more overlapping pairs. This will happen eventually because all overlaps are resolved by moving both rectangles in opposing directions. This will broadcast overlaps toward outer rectangles of a cluttered area or even the whole input set and terminate when those can be moved into empty drawing area. The theoretical runtime of an iteration is dominated by finding the k intersecting pairs, which can be done in $\mathcal{O}(|V| \cdot \log |V| + k)$ [BW80].

Identical ranks In the current implementation the algorithm terminates with an error message if it encounters two rectangles with identical ranks, because in this case the overlap cannot be removed without violating the orthogonal order. Rectangles get the same ranks if they have the same coordinates. In practice, this would correspond to multiple items of metadata that belong to the exact same geographical location. One way to deal with this problem is to aggregate such rectangles to a single, larger one, possibly containing several sub-rectangles. In our application this would make sense, since they are all describing the same site. An alternative and more general solution would be to apply a very small random displacement to the input data, to remove any occurrence of identical coordinates. The datasets we used for our evaluation did not contain duplicate coordinates, so neither of these methods were used.

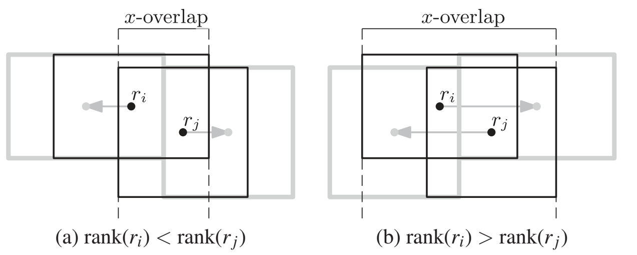


Figure 2: *Overlap computation for correct (a) and reversed (b) orthogonal order in the x-dimension*

4. Method REPAIRORDER

Although REARRANGE considers the orthogonal order when it removes the overlap between a pair of rectangles, it can still introduce order violations. The rectangles from the pair under consideration can be pushed past other rectangles in the layout. Therefore, we introduce a method to repair the orthogonal order after each iteration.

The REPAIRORDER method is a modified version of MERGESORT. Our strategy is similar to the *project* procedure Dwyer et al. added to *stress majorization* in [DKM06], but finds and resolves the inversions while sorting. It takes as input a list $L = (r_1, \ldots, r_n)$ of vertices, dimension $d \in \{x, y\}$ for which the order should be repaired, and the markers *left* and *right* for the sublist that should be considered in the current call. In the initial call the rectangles in L are sorted by their current coordinate in dimension d. The algorithm performs MERGESORT on this list of vertices, with two additions when merging a left with a right list:

- Whenever an element e_r in the right list has a lower rank than an element e_l in the left list, we do not only swap the vertices in the list, we also recompute their coordinates to satisfy the orthogonal order constraints: We compute the average of their coordinates (in dimension d) and place both of them on this coordinate.
- If in the merging step we encounter two vertices with the same rank, we collect all vertices of this rank, compute the average of their coordinates (in dimension d), and place them all on this average coordinate.

We extend both REARRANGE and the popular overlap removal algorithm PRISM with the REPAIRORDER method to resolve order violations after each iteration. Repairing the order might introduce new overlaps, which will be treated in the next iteration.

The runtime of a REPAIRORDER iteration is dominated by resolving the inversions while sorting, which is the runtime of

Algorithm 2: REPAIRORDER

```
input : List $L = (r_1, \dots, r_n)$ of rectangles, dimension
        $d \in \{x, y\}$ for which the order has to be repaired,
        markers *left* and *right* defining the sublist that should
        be considered.
output: The same list of vertices with adjusted coordinates.
1  Let $c_i$ be the coordinate and $rank(r_i)$ the rank of rectangle
     $R[i] = r_i$ in the given dimension $d$;
2  if $left < right$ then
3      $mid \leftarrow \lfloor (left + right - 1)/2 \rfloor$;
4      REPAIRORDER($L, d, left, mid$);
5      REPAIRORDER($L, d, mid + 1, right$);
6      $i \leftarrow left$;   $j \leftarrow mid + 1$;   $k \leftarrow left$;
7      while $i \leq mid$ and $j \leq right$ do
8          if $rank(r_i) < rank(r_j)$ then
9              $r_i$ is first in the order, no action required;
10             $L'[k] \leftarrow L[i]$;   $i{+}{+}$;   $k{+}{+}$;
11         else
12             cr $\leftarrow rank(r_i)$; group $\leftarrow \emptyset$;
13             while $i \leq mid$ do
14                 group.add($r_i$);
15                 $i{+}{+}$;
16             while $rank(r_j) = cr$ do
17                 group.add($r_j$);
18                 $L'[k] \leftarrow L[j]$;   $j{+}{+}$;   $k{+}{+}$;
19             $c_{avg} \leftarrow$ average $c_g$ of all $r_g \in group$;
20             forall $r_g \in group$ do
21                 $c_g \leftarrow c_{avg}$;
22     for $h \leftarrow i, \dots, mid$ do
23         $L[k + h - i] \leftarrow L[h]$;
24     for $h \leftarrow left, \dots, k - 1$ do
25         $L[h] \leftarrow L'[h]$;
```

MERGESORT plus the number i of inversions that have to be resolved, hence $\mathcal{O}(n \log n + i)$.

To verify that the algorithm still terminates we have to take a closer look at what happens when an inversion is repaired. When an overlap removal step caused an order violation, repairing this violation will at most push the rectangle back to its previous position. This might re-introduce some overlap, but since the overlap was removed by moving both rectangles, it is now smaller than before. Again, the overlap is propagated to the outer borders of the more cluttered areas or the entire pointset, where it can be resolved without introducing new violations.

5. Evaluation

We evaluate our heuristic REARRANGE in comparison to PRISM, and we extend both algorithms with the order repair method (PRISM+OR and REARRANGE+OR) to guarantee preservation of the orthogonal order. Both algorithms and the extension with order repair were implemented in Java, building on the framework of the network visualization tool Visone (`www.visone.info`). Experiments were executed on a 64-bit desktop PC with an Intel Core i7-4790 CPU (3.60 GHz, 8 cores, 8 MB cache) and 16 GB RAM. We use the following datasets to evaluate the algorithms.

Geographical data We have one real-life dataset, a collection of 69 cultural heritage sites on the island of St. Kitts obtained through a crowdsourcing project [HSPT]. Each site should be represented by a card with metadata, which includes site name, type of heritage, and optionally a picture and/or a description. These cards determine the size of the rectangles. To test our methods more extensively, we extracted additional data from OpenStreetMap. To approximate archaeological site data on islands, we use the locations of important places such as churches, schools, and museums on several islands in Central America. We have a total of 134 such datasets, each containing between 20 and 200 site locations. Since the hypothetical representations of metadata for these sites could have any shape or size, we model them by generating rectangles with randomly chosen width and height. The rectangles have an aspect ratio between 1:4 and 4:1, and their average size depends on the area covered by the pointset and the number of points. The middle column in Figure 6 shows examples of this type of input. We generated three sets of random rectangles for each set of site locations. For comparison, we also created one consisting of uniform squares and one consisting of uniform rectangles with ratio 3:1.

Synthetic data For graph-related applications of overlap removal nodes are often assumed to be of uniform size and shape, when textual labels are expected they tend to all be wide rectangles. For our application, however, we expect varying shapes of metadata-symbols. To investigate the influence of label shapes on the performance of the methods in more detail, we generated a series of synthetic datasets with varying labels. The initial layout always consists of normally distributed clusters spread around a circle, 100

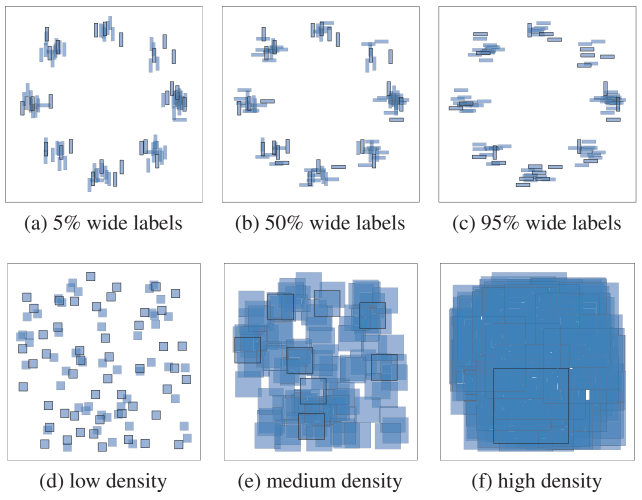

(a) 5% wide labels (b) 50% wide labels (c) 95% wide labels
(d) low density (e) medium density (f) high density

Figure 3: *Examples of synthetic test data with varying label shapes (a-c) and density (d-f)*

nodes in total. We start with only tall labels (rectangles with aspect ratio 1:4), and replace them with wide labels (rectangles with aspect ratio 4:1) one by one, until all labels are wide. For each tall-to-wide ratio we run the algorithms and compare the results. For the application of labeling geographic maps we expect clustered datasets with relatively small amounts of overlap and a lot of empty space. To test what happens when the data is homogenous and dense, we created a series of synthetic datasets with increasing label size. The layout is a uniform random distribution of 100 points, the labels are equal-sized squares. We run the algorithm with the different label sizes and compare the results. Both types of synthetic data are illustrated in Figure 3.

5.1. Quality metrics

To compare the quality of the outputs of the different algorithms, we consider the following measures:

- Our objective is to keep the total displacement of the rectangles small, so we measure the total displacement of the rectangles relative to their initial positions.
- We are also interested in how well the shape of the input is maintained. If the original graph is just scaled, shifted, or rotated as a whole, we would consider the result to have the same shape as before. The neglection of such transformations is achieved by finding the optimal scaling, rotation/reflection, and shifting such that the displacement of the vertices is minimized. This is known as the Procrustes transformation [BG05]. We use the `protest` method in R [OBF*16] to perform a Procrustes analysis matching the result of each method to the input data, and compute the Procrustes correlation between the initial positions and the transformed resulting layout. The correlation is a value between 0 and 1, where a value close to 1 indicates that two shapes are similar.
- When we run the algorithms without the ORDERREPAIR method we also count the number of order violations in the final layout.
- We also record and compare the runtimes of the algorithms.

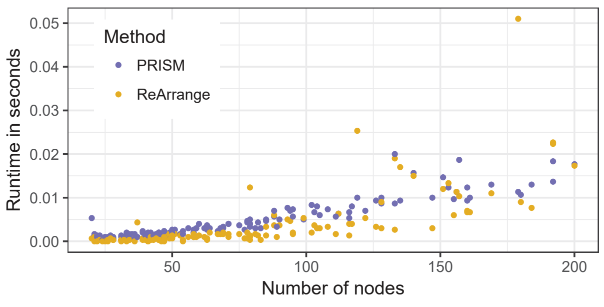
	PRISM	ReArrange
mean displacement	72.63	50.37
% won on displacement	0	100
mean procrustes	0.9920	0.9959
% won on procrustes	0.8	99.2
mean order violations	3.15	3.02
% won order violations	44.3	55.7
mean runtime	0.0047	0.0035
% won on runtime	8.4	88.5

	PRISM+OR	ReArrange+OR
mean displacement	135.0	115.33
% won on displacement	14.0	86.0
mean procrustes	0.9802	0.9847
% won on procrustes	12.5	87.5
mean runtime	1.2078	0.0815
% won on runtime	4.6	95.4

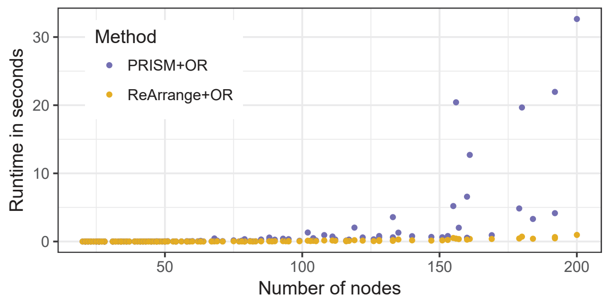
Table 1: *Summary of the results for all four methods applied to the geographical datasets with randomized label shapes*

5.2. Results

Table 1 summarizes the results for all four methods applied to the geographic datasets with varying label shapes. The percentage of won cases indicated in this table is the percentage of datasets for which this method scored better for this measure than the competing method. Figure 5 shows the results for displacement, Procrustes correlation and order violations in more detail. Each dot represents one of the geographic datasets. The average displacement per node, Procrustes correlation, or number of violations that the two methods without or with order preservation achieved on this dataset are plotted against each other. A dot above the diagonal line indicates a case where PRISM performed better than REARRANGE, a dot below the diagonal shows a case where REARRANGE outperformed PRISM. These plots show that without order repair, REARRANGE always results in a smaller average displacement than PRISM (Fig. 5a) and with only two exceptions also achieves a higher Procrustes correlation (Fig. 5b), meaning better shape preservation. When we extend both approaches with our ORDERREPAIR method the number of exceptions increases (Fig. 5d,e), but for both displacement and Procrustes correlation REARRANGE still performs better in over 85% of the cases (Table 1). Figure 5c shows the number of orthogonal order violations in the results of both methods without order repair. Here the performance of both methods is comparable for the smaller datasets (blue dots), but especially for the larger datasets (red dots) REARRANGE tends to cause fewer order violations. Overall, REARRANGE caused fewer violations in roughly 55% of the cases.



(a) Runtime comparison for algorithms without order repair



(b) Runtime comparison for algorithms with order repair

Figure 4: *Comparison of runtime in seconds, averaged over three runs for each algorithm. Both with and without order repair, REARRANGE is faster in the vast majority of the cases.*

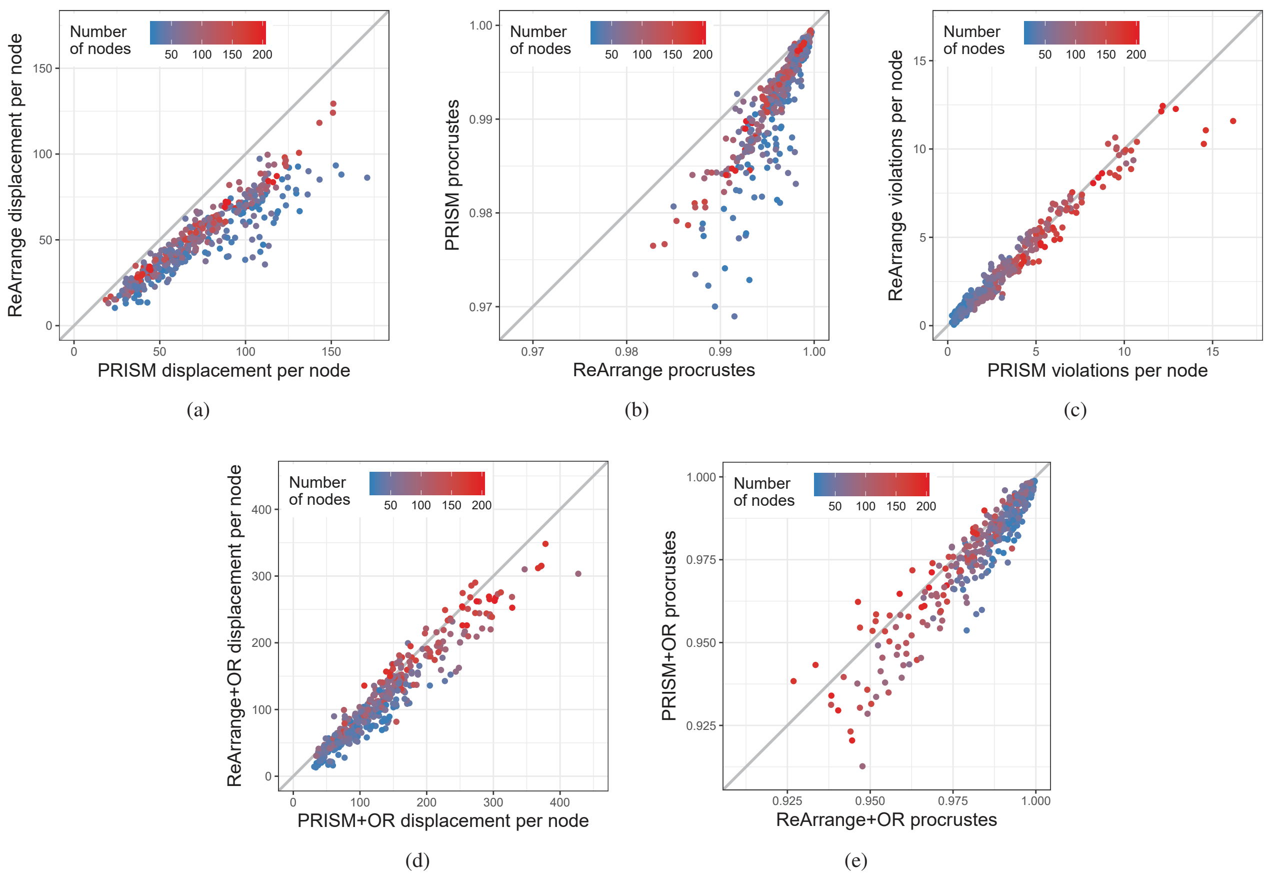
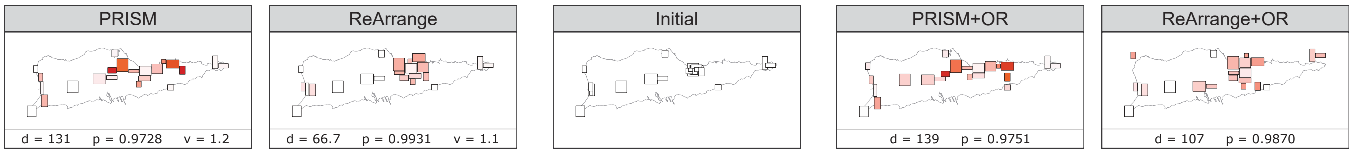


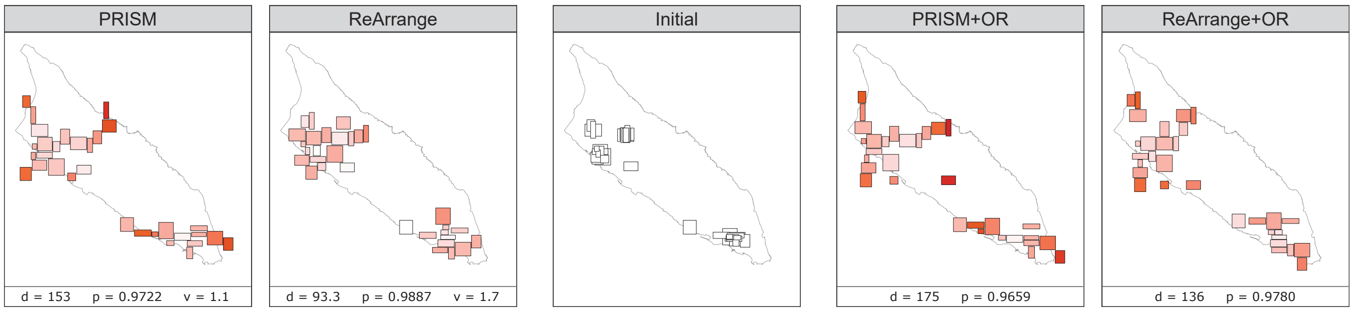
Figure 5: *Displacement (a), Procrustes correlation (b) and order violations (c) for PRISM versus* REARRANGE *and displacement (d) and Procrustes correlation (e) for PRISM with order repair versus* REARRANGE *with order repair. Each dot represents one test instance, the color indicates the number of nodes in this instance. Points below the diagonal represent instances where* REARRANGE *outperformed PRISM.*

Figure 4 shows the runtimes of all four methods. Without order repair REARRANGE tends to be slightly faster than PRISM, but the differences are quite small and for the larger instances there are a few outliers. When we add order repair to both algorithms the differences in runtime become much larger and REARRANGE is faster in over 95% of the cases. Figure 4b shows that especially for larger instances, PRISM+OR often takes extremely long. This can once again be explained by the fact that PRISM introduces more order violations in its overlap removal step. When many violations have to be repaired, many movements have to be (partially) undone, and this causes the combined method to converge much slower than without order repair.

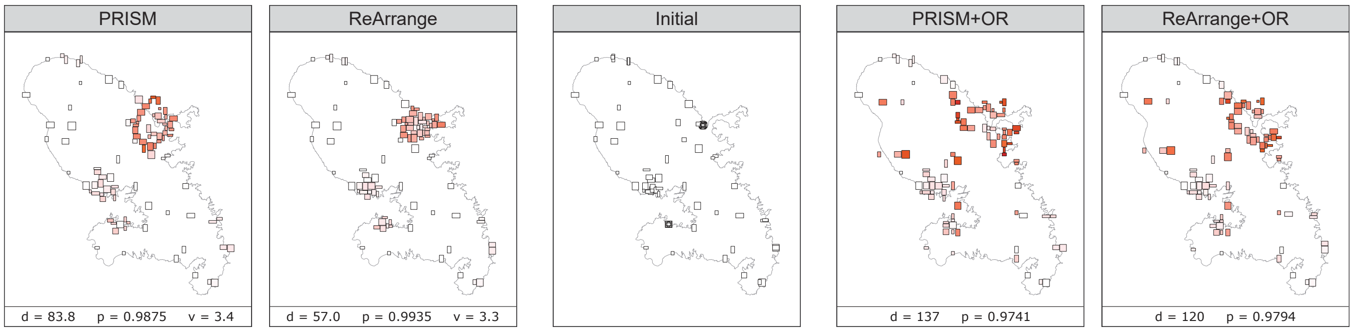
Figure 6 shows four example cases that help explain why REARRANGE often performs better than PRISM. For each example the initial positions (middle), the resulting layouts for both methods without order preservation (left), and the resulting layouts for both methods with order violation (right) are shown. The color of each rectangle corresponds to its displacement, with white representing no displacement and dark red representing maximal displacement. As explained in Section 2, PRISM removes overlap along the edges of the proximity graph rather than in the direction of minimum displacement, which can lead to unnecessarily large displacements. This is most clearly illustrated by the cluster of three rectangles on the left of Figure 6a. A similar problem occurs for larger dense clusters, as shown in Figure 6b. PRISM pushes multiple rectangles along a ray out of the center, introducing some large displacements. REARRANGE keeps the rectangles closer together by removing overlap for each pair in the direction of minimum displacement. As a result of moving rectangles in dense clusters radially outwards PRISM sometimes creates rings of rectangles with empty centers, which also results in a large displacement. This case is illustrated in Figure 6c. As explained above, PRISM tends to cause more order violations than REARRANGE, especially in larger graphs. When we extend both methods with order repair this becomes a disadvantage for PRISM. As shown in Figure 6d, many rectangles end up on one line. This is a result of the order repair method that is executed after each iteration. Because PRISM causes more violations, more of them have to be repaired, which is done by putting the violating rectangles on one line. All of the issues described here have a negative influence on the displacement as well as the Procrustes correlation for the PRISM+OR results.



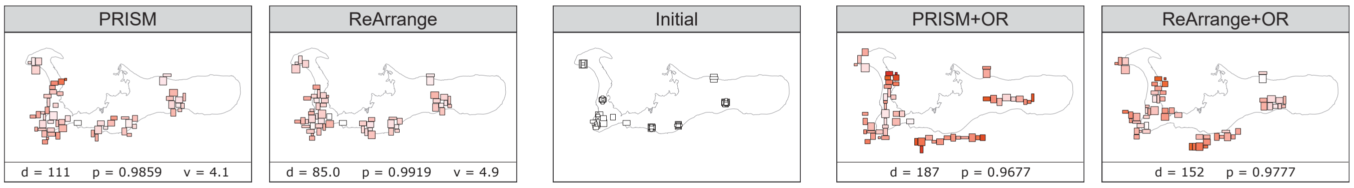
(a) Parks on Saint Croix



(b) Schools on Aruba



(c) Churches on Martinique



(d) Schools on Grand Cayman

Figure 6: *Layout results for four examples. The initial layout is shown in the middle, the results of the two methods without order repair on the left, and the results of both methods with order repair on the right. The rectangles are colored by their displacement. The lighter-colored rectangles in the* REARRANGE *layouts indicate that* REARRANGE*(+OR) results in a smaller displacement than PRISM(+OR). The numbers at the bottom indicate the average displacement (d), Procrustes correlation (p) and average number of violations (v).*

Label shapes To test the influence of the shape of the labels, we also applied the algorithms to the geographic datasets with uniform labels. For uniform squares the results are very similar to those with random rectangles, but when we use uniform wide rectangles (ratio 3:1) we do see some changes. In terms of displacement and Procrustes correlation both methods perform worse with the wide rectangles. The win-percentage of REARRANGE drops to 78% of the cases for displacement and 64% for Procrustes correlation. To study this effect in more detail, we apply all four methods to the synthetic data with varying label shapes. Figure 7 shows the results in terms of displacement and Procrustes correlation. The percentage of wide labels is plotted along the x-axis, the remaining labels are tall. The performance of REARRANGE is quite stable for varying shape distributions, but it obtains slightly better results when both shapes occur equally often. PRISM on the other hand performs best with uniform labels. With respect to displacement REARRANGE remains the clear winner in any case, but when almost all of the labels are wide rectangles there is not much difference between the two algorithms regarding the Procrustes correlation. For the methods with order repair we observe a similar pattern but with more variation in the results. REARRANGE performs better on average, but the difference with PRISM is largest for input with varying

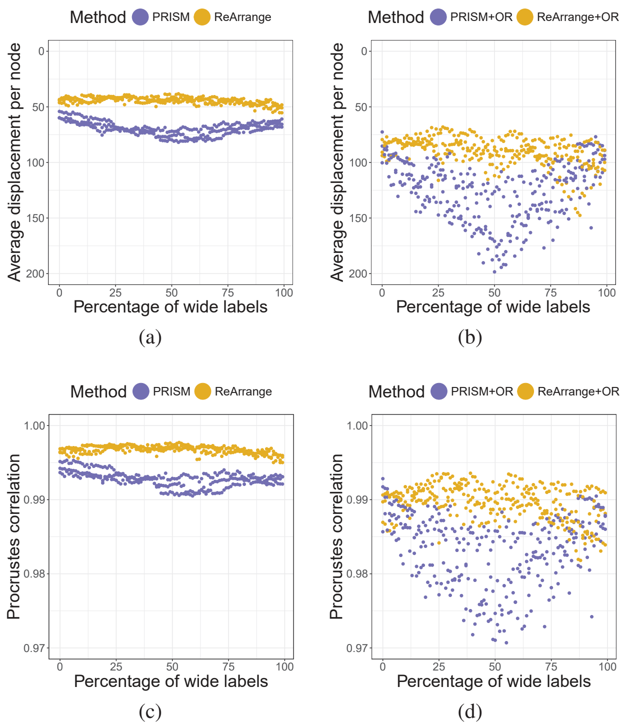


Figure 7: *Variation in results depending on label shapes. The performance of* ReArrange *remains similar for varying label shape distributions, PRISM clearly performs better with uniform labels than with a mix of tall and wide rectangles.*

label shapes. This can be explained by the fact that our heuristic decides in which direction an overlap should be removed based on in which dimension the overlap is smaller. When all labels are very wide, the overlap in y-direction is much more likely to be smaller and the image will be stretched in one direction. This has a negative influence on the Procrustes correlation.

Density Another factor that could influence how well the algorithms perform is the density of the input. We tested series of uniform-randomly distributed datasets with equal-sized squares, where we increased the density by scaling up the labels in each step. In all cases ReArrange achieves a smaller displacement, and difference with PRISM gets larger as the density of the input increases. Figure 8 shows the results for Procrustes correlation and the number of order violations. For both these measures ReArrange performs better than PRISM for low-density input, but for high-density input it is the other way around. When we add order repair to the algorithms, ReArrange still achieves a smaller displacement in over 80% of the cases. Both algorithms perform worse regarding Procrustes correlation as the density increases, but this has no clear effect on which one performs better.

Application example In our application of visualizing meta-data of archaeological sites, varying label shapes and relatively low density are to be expected due to the nature of the data. Our real-life dataset of cultural heritage sites on the island of Saint Kitts is one

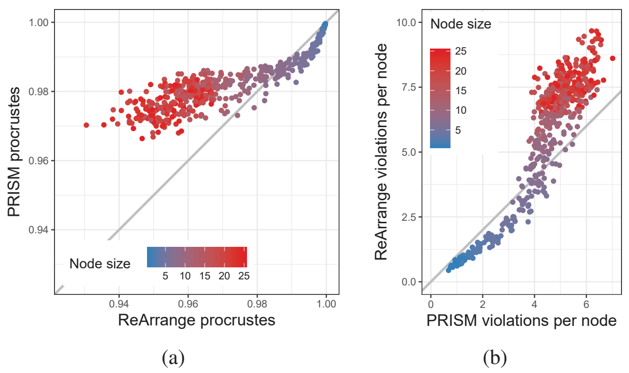


Figure 8: *Procrustes correlation (a) and order violations (b) for PRISM versus* ReArrange*, showing the variation in results depending on density. Each dot represents one test instance, the color indicates the node size in this instance. Points below the diagonal represent instances where* ReArrange *outperformed PRISM.*

example of this. Each site is represented by a card, but since some elements are missing for some cards they have varying dimensions. Figure 9 shows an overlap-free layout for this dataset, computed with ReArrange+OR.

6. Conclusion

We presented a new heuristic for the minimum-displacement overlap removal problem. In addition, we proposed an order repair method that resolves orthogonal order violations after every iteration of overlap removal. We extended our own heuristic ReArrange and the commonly used overlap removal algorithm PRISM with this repair method, and compared the results. We evaluated both approaches with and without order repair using various metrics. The experiments show that our new heuristic approach results in the smallest displacement for almost all of the data sets and maintains the shape of the input better for most of them. This makes it especially useful for applications which require a non-overlapping placement of objects close to desired (geographical) positions, like symbol maps of archeological sites. Furthermore, our experiments indicate that our heuristic performs especially well when the input contains labels of varying sizes.

In future work we aim to further improve our heuristic algorithm, and to explore how well it performs on non-geographical data. It would also be interesting to study the influence of different label shapes in more detail. Furthermore, would like to generate test data sets that have different characteristics according to known spatial statistics to understand the influence of different point configurations better. Another potential research direction would be to look into trade-offs between displacement and other possibilities to remove overlap between rectangles, such as small changes in their aspect-ratio. Such changes could reduce the displacement needed to obtain a disjoint layout. Finally, it would be interesting to also compare our approach to other overlap removal methods used in practice.

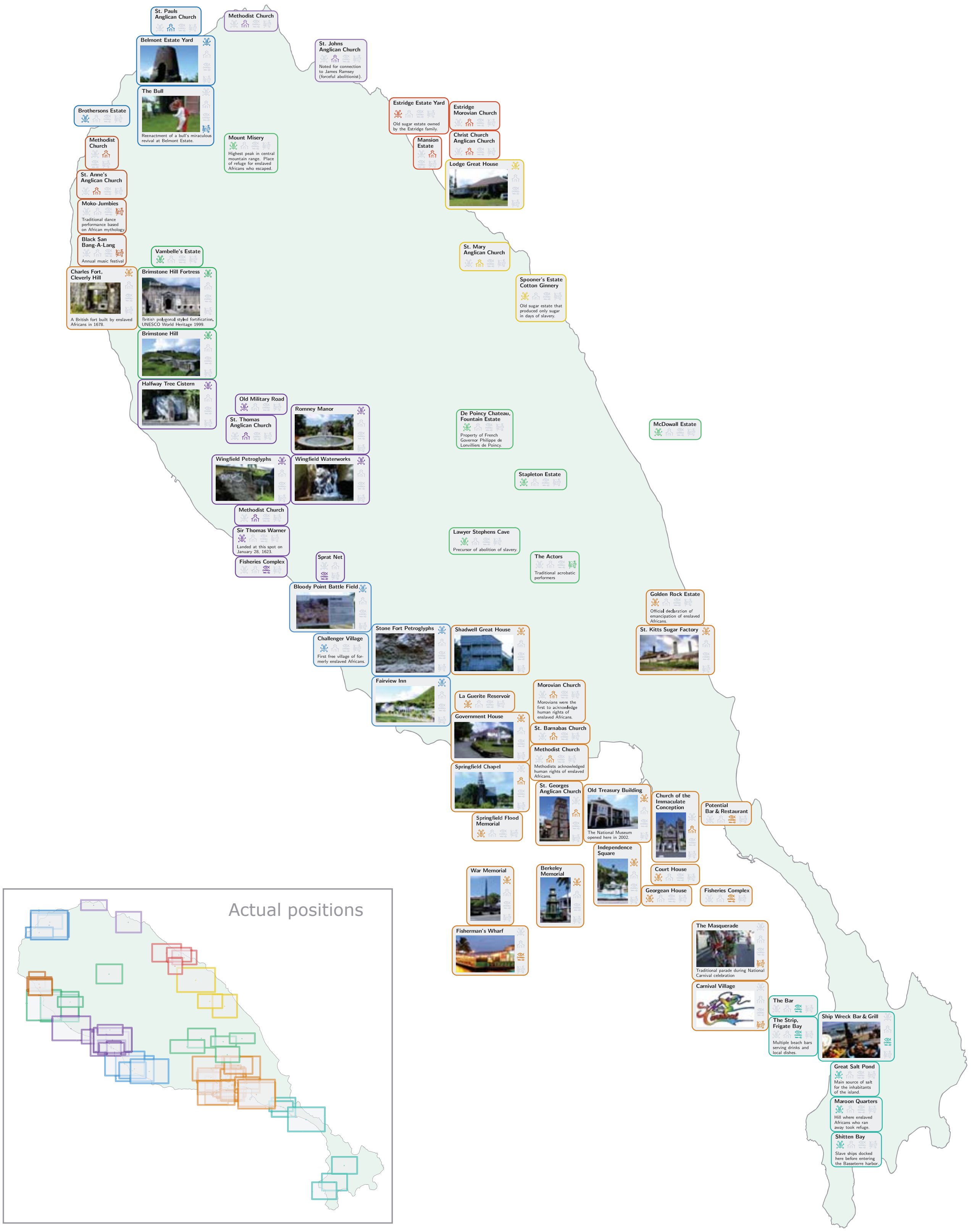

St. Pauls Anglican Church
Methodist Church
Belmont Estate Yard
St. Johns Anglican Church
Noted for connection to James Ramsey (forceful abolitionist).
The Bull
Reenactment of a bull's miraculous revival at Belmont Estate.
Brothersons Estate
Estridge Estate Yard
Old sugar estate owned by the Estridge family.
Estridge Moravian Church
Methodist Church
Mount Misery
Highest peak in central mountain range. Place of refuge for enslaved Africans who escaped.
Mansion Estate
Christ Church Anglican Church
Lodge Great House
St. Anne's Anglican Church
Moko-Jumbies
Traditional dance performance based on African mythology.
Black San Bang-A-Lang
Annual music festival
Vambelle's Estate
St. Mary Anglican Church
Charles Fort, Cleverly Hill
A British fort built by enslaved Africans in 1678.
Brimstone Hill Fortress
British polygonal styled fortification, UNESCO World Heritage 1999.
Spooner's Estate Cotton Ginnery
Old sugar estate that produced only sugar in days of slavery.
Brimstone Hill
Halfway Tree Cistern
Old Military Road
St. Thomas Anglican Church
Romney Manor
De Poincy Chateau, Fountain Estate
Property of French Governor Philippe de Lonvilliers de Poincy.
McDowall Estate
Wingfield Petroglyphs
Wingfield Waterworks
Stapleton Estate
Methodist Church
Sir Thomas Warner
Landed at this spot on January 28, 1623.
Lawyer Stephens Cave
Precursor of abolition of slavery.
Fisheries Complex
Sprat Net
The Actors
Traditional acrobatic performers
Bloody Point Battle Field
Golden Rock Estate
Official declaration of emancipation of enslaved Africans.
Stone Fort Petroglyphs
Shadwell Great House
St. Kitts Sugar Factory
Challenger Village
First free village of formerly enslaved Africans.
Fairview Inn
Morovian Church
Moravians were the first to acknowledge human rights of enslaved Africans.
La Guerite Reservoir
Government House
St. Barnabas Church
Methodist Church
Methodists acknowledged human rights of enslaved Africans.
Springfield Chapel
St. Georges Anglican Church
Old Treasury Building
The National Museum opened here in 2002.
Church of the Immaculate Conception
Potential Bar & Restaurant
Springfield Flood Memorial
Independence Square
War Memorial
Berkeley Memorial
Court House
Georgean House
Fisheries Complex
Fisherman's Wharf
The Masquerade
Traditional parade during National Carnival celebration
Carnival Village
The Bar
The Strip, Frigate Bay
Multiple beach bars serving drinks and local dishes.
Ship Wreck Bar & Grill
Great Salt Pond
Main source of salt for the inhabitants of the island.
Maroon Quarters
Hill where enslaved Africans who ran away took refuge.
Shitten Bay
Slave ships docked here before entering the Basseterre harbor.
Actual positions

Figure 9: *Layout result for real-world data: meta-data cards representing culture and heritage sites on the island of Saint Kitts. This order-preserving layout was computed using* ReArrange *extended with order repair. A map of the island is projected in the background with the same distortion as the bounding box of the point set. The actual positions are shown in the lower left corner.*

Acknowledgments

The authors wish to thank Tom Brughmans for his very helpful insights from the view of an archeologist. This research is part of the projects NEXUS 1492, funded by the European Research Council (ERC) under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement n^{o} 319209, and CARIB, funded by the HERA Joint Research Programme which is co-funded by AHRC, AKA, BMBF via PT-DLR, DASTI, ETAG, FCT, FNR, FNRS, FWF, FWO, HAZU, IRC, LMT, MHEST, NWO, NCN, RANNCDS, RCN, VR and The European Community FP7 2007-2013, under the Socio-economic Sciences and Humanities Programme. We also acknowledge Deutsche Forschungsgemeinschaft (DFG) for support of project B02 in SFB/TRR 161.

References

[BG05] Borg I., Groenen P. J.: *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media, 2005. 6

[Bri11] Bright A. J.: *Blood is thicker than water: Amerindian intra-and inter-insular relationships and social organization in the pre-Colonial Windward Islands*. Sidestone Press, 2011, ch. Windward islands recalibrated settlement sequence and ceramic age settlement system, pp. 101–140. 2

[BW80] Bentley J. L., Wood D.: An optimal worst case algorithm for reporting intersections of rectangles. *IEEE Transactions on Computers C-29*, 7 (July 1980), 571–577. doi:10.1109/TC.1980.1675628. 4

[CL06] Conolly J., Lake M.: *Geographical information systems in archaeology*. Cambridge University Press, 2006, ch. Maps and Digital Cartography, pp. 263–279. 1, 2

[DKM06] Dwyer T., Koren Y., Marriott K.: Stress majorization with orthogonal ordering constraints. In *Graph Drawing: 13th International Symposium, GD 2005, Limerick, Ireland, September 12-14, 2005. Revised Papers* (Berlin, Heidelberg, 2006), Healy P., Nikolov N. S., (Eds.), Springer Berlin Heidelberg, pp. 141–152. URL: http://dx.doi.org/10.1007/11618058_14, doi:10.1007/11618058_14. 4

[DMS06] Dwyer T., Marriott K., Stuckey P. J.: Fast node overlap removal. In *Graph Drawing: 13th International Symposium, GD 2005, Limerick, Ireland, September 12-14, 2005. Revised Papers* (Berlin, Heidelberg, 2006), Healy P., Nikolov N. S., (Eds.), Springer Berlin Heidelberg, pp. 153–164. URL: http://dx.doi.org/10.1007/11618058_15, doi:10.1007/11618058_15. 3

[Dor96] Dorling D.: Area cartograms: their use and creation. In *Concepts and techniques in modern geography* (1996), vol. 59. 3

[GF15] Golitko M., Feinman G. M.: Procurement and distribution of pre-hispanic mesoamerican obsidian 900 bc–ad 1520: a social network analysis. *Journal of Archaeological Method and Theory 22*, 1 (2015), 206–247. URL: http://dx.doi.org/10.1007/s10816-014-9211-1, doi:10.1007/s10816-014-9211-1. 2

[GH09] Gansner E. R., Hu Y.: Efficient node overlap removal using a proximity stress model. In *Graph Drawing: 16th International Symposium, GD 2008, Heraklion, Crete, Greece, September 21-24, 2008. Revised Papers* (Berlin, Heidelberg, 2009), Tollis I. G., Patrignani M., (Eds.), Springer Berlin Heidelberg, pp. 206–217. URL: http://dx.doi.org/10.1007/978-3-642-00219-9_20, doi:10.1007/978-3-642-00219-9_20. 3

[HIMF98] Hayashi K., Inoue M., Masuzawa T., Fujiwara H.: A layout adjustment problem for disjoint rectangles preserving orthogonal order. In *Graph Drawing* (1998), Springer, pp. 183–197. 2, 3

[HL03] Huang X., Lai W.: Force-transfer : A new approach to removing overlapping nodes in graph layout. In *Twenty-Sixth Australasian Computer Science Conference (ACSC2003)* (Adelaide, Australia, 2003), Oudshoorn M. J., (Ed.), vol. 16 of *CRPIT*, ACS, pp. 349–358. 3

[HM97] He W., Marriott K.: Constrained graph layout. In *Graph Drawing: Symposium on Graph Drawing, GD '96 Berkeley, California, USA, September 18–20, 1996 Proceedings* (Berlin, Heidelberg, 1997), North S., (Ed.), Springer Berlin Heidelberg, pp. 217–232. URL: http://dx.doi.org/10.1007/3-540-62495-3_50, doi:10.1007/3-540-62495-3_50. 3

[HM15] Homsey-Messer L.: Revisiting the role of caves and rockshelters in the hunter-gatherer taskscape of the archaic midsouth. *American Antiquity 80*, 2 (2015), 332–352. URL: http://www.ingentaconnect.com/content/saa/aa/2015/00000080/00000002/art00006, doi:doi:10.7183/0002-7316.80.2.332. 2

[HSPT] Habiba, Stancioff E., Philips M., Thatcher M. R.: Culturesnaps. http://www.culturesnaps.kn. 5

[HWAT13] Hirono D., Wu H. Y., Arikawa M., Takahashi S.: Constrained optimization for disoccluding geographic landmarks in 3d urban maps. In *2013 IEEE Pacific Visualization Symposium (PacificVis)* (2013), pp. 17–24. doi:10.1109/PacificVis.2013.6596123. 3

[KLKS10] Koh K., Lee B., Kim B., Seo J.: Maniwordle: providing flexible control over wordle. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 1190–1197. 3

[LEN05] Li W., Eades P., Nikolov N.: Using spring algorithms to remove node overlapping. In *Proceedings of the 2005 Asia-Pacific Symposium on Information Visualisation - Volume 45* (Darlinghurst, Australia, Australia, 2005), APVis '05, Australian Computer Society, Inc., pp. 131–140. URL: http://dl.acm.org/citation.cfm?id=1082315.1082334. 3

[Lyo92] Lyons K. A.: Cluster busting in anchored graph drawing. In *Proceedings of the 1992 Conference of the Centre for Advanced Studies on Collaborative Research - Volume 2* (1992), CASCON '92, IBM Press, pp. 327–337. URL: http://dl.acm.org/citation.cfm?id=962260.962284. 3

[MELS95] Misue K., Eades P., Lai W., Sugiyama K.: Layout adjustment and the mental map. *Journal of visual languages and computing 6*, 2 (1995), 183–210. 2

[Mor15] Morin J.: Near-infrared spectrometry of stone celts in precontact british columbia, canada. *American Antiquity 80*, 3 (2015), 530–547. URL: http://www.ingentaconnect.com/content/saa/aa/2015/00000080/00000003/art00006, doi:doi:10.7183/0002-7316.80.3.530. 2

[MSTH03] Marriott K., Stuckey P., Tam V., He W.: Removing node overlapping in graph layout using constrained optimization. *Constraints 8*, 2 (2003), 143–171. URL: http://dx.doi.org/10.1023/A:1022371615202, doi:10.1023/A:1022371615202. 3

[OBF*16] Oksanen J., Blanchet F. G., Friendly M., Kindt R., Legendre P., McGlinn D., Minchin P. R., O'Hara R. B., Simpson G. L., Solymos P., Henry M., Stevens H., Szoecs E., Wagner H.: *vegan: Community Ecology Package*, 2016. R package version 2.4-1. URL: http://CRAN.R-project.org/package=vegan. 6

[SSS*12] Strobelt H., Spicker M., Stoffel A., Keim D., Deussen O.: Rolled-out wordles: A heuristic method for overlap removal of 2d data representatives. *Computer Graphics Forum 31*, 3pt3 (2012), 1135–1144. URL: http://dx.doi.org/10.1111/j.1467-8659.2012.03106.x, doi:10.1111/j.1467-8659.2012.03106.x. 3

[VWF09] Viegas F. B., Wattenberg M., Feinberg J.: Participatory visualization with wordle. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (Nov. 2009), 1137–1144. URL: http://dx.doi.org/10.1109/TVCG.2009.171, doi:10.1109/TVCG.2009.171. 3