Universität Konstanz

Chair of Computer Graphics and Media Design
Department of Computer Science

Bachelor Thesis

Overlap Removal Methods for Data Projection Algorithms

Überlappungsbereinigung für Algorithmen zur Datenprojektion

for obtaining the academic degree
Bachelor of Science (B.Sc.)

Field of study: Information Engineering
Focus: Information Visualization

by

Marc Spicker
(01/691558)

First advisor: Prof. Dr. Oliver Deussen
Second advisor: Prof. Dr. Daniel Keim

Konstanz, September 30, 2011

Declaration of Authorship

The author of this work hereby declares that

- the present work is the result of his own work, without help from others and without using anything other than the named sources and aids;

- the texts, illustrations and/or ideas taken directly or indirectly from other sources (including electronic resources) have without exception been acknowledged and have been referenced in accordance with academic standards.

The Author wants to gratefully acknowledge supervision and guidance he has received from Hendrik Strobelt.

Konstanz, September 30, 2011 Marc Spicker

Abstract

Projection algorithms map high dimensional data points to lower dimensions. However, when adding arbitrary shaped objects as representatives for these data points, they may intersect. The positions of these representatives have to be modified in order to remove existing overlaps. There are multiple algorithms designed to deal with this layout adjustment problem, which lead to very different results. These adjustment strategies are evaluated according to different measures for comparison: euclidean distance, area occupancy, and orthogonal ordering. Four common overlap removal algorithms are adopted and evaluated. Taking drawbacks of the observed algorithms into account, a new overlap removal algorithm has been developed. Finally, measures underlay improvements of the new algorithm.

Contents

1 Introduction **1**

2 Overlap Removal Methods **3**
2.1 Box2D Physics Engine . . . 3
2.1.1 Non-granular Objects . . . 4
2.1.2 Granular Objects . . . 5
2.2 Wordle (ManiWordle) . . . 6
2.3 Voronoi . . . 9
2.4 VPSC . . . 13

3 Evaluation **15**
3.1 Methods . . . 15
3.1.1 Euclidean Distance Measure . . . 15
3.1.2 Area Occupancy . . . 16
3.1.3 Orthogonal Ordering . . . 16

4 Results **18**
4.1 Dataset: Separated Cluster . . . 18
4.2 Dataset: Overlapping Cluster . . . 21
4.3 Dataset: Uniform Distribution . . . 23
4.4 Intermediate Conclusion . . . 26

5 Improvements **27**
5.1 Scanline Wordle . . . 27
5.1.1 Algorithm . . . 27
5.1.2 Results . . . 30
5.1.3 Scanline Wordle & Real World Data . . . 32

6 Conclusion **46**

7 Future Work **47**

8 Bibliography **50**

1 Introduction

In visualizations, arbitrary shaped entities are used at meaningful positions to convey informations to a user. Maps for example have cities denoted by points or other symbols, often dependent on their number of inhabitants, at their geographic locations. Additional visual representatives (like the name of the city) can be drawn next to it in order to add even more information to the visualization.

When simply drawing the visual representatives of each entity at the given position, overlap will occur. With overlaps in a visualization, informations are partly occluded. This makes it less useful and possibly even less visually appealing: The visualization fails to convey informations efficiently. Any layout containing entities with arbitrary shapes can suffer from this issue. The same problem exists in the field of graph drawing and has been examined quite heavily in the past decades [7]. In order to remove all overlaps, layout adjustment algorithms change the positions of the entities so that their visual representatives do not intersect any longer. When adjusting the layout, it is important to retain the informations in the visualization. The easiest way to remove all overlaps is to uniformly scale up the layout with a sufficient large scaling factor. But this turns out to be inappropriate for most use cases, because the resulting layouts grow very large. Therefore, other algorithms are needed to overcome this issue.

Related work has mostly been done in the field of graph drawing, but the given problems can easily be transferred into graph terms: Entities can be seen as nodes and every distance between two nodes as an edge, creating an compete graph.

There are two main classes of overlap removal algorithms: one directly addresses the problem of different node sizes and shapes in the initial layout [11], and others that adjust an initial layout, in which nodes are seen as points. We will focus on the latter. Multiple algorithms exist that target overlap removal by adjusting a initial layout without simply scaling the initial layout: *Spring algorithms* [17] view nodes as rings and edges as springs which apply forces to two rings. The resulting, overlapping free layout is the minimal energy state. *Force based* algorithms use forces similar to the spring algorithms, but differ at the positioning and the strength of the forces. The *Force Scan (FS)* algorithm [21] scans the initial layout horizontally and vertically and moves the overlapping nodes apart along a thought line which connects their centers while preserving the orthogonal ordering. The modified variant of this algorithm is the *FS'* algorithm [14] which tries to produce more compact layouts. A quite similar algorithm, the *Force Transfer algorithm* [15] applies forces to the "transfer neighbor nodes" of a node, which are all the nodes in overlap-removal direction of a certain node. Other force based algorithms include *cluster busting* [19], such as *Voronoi Cluster busting*. They try to even the distribution of nodes, because clusters of nodes are areas with high overlap probabilities. Another class of overlap removal algorithms

are *constrained optimization* algorithms [20, 10]. They generate "separation constraints" that ensure non-overlapping and find a optimized solution via quadratic programming which solves all these constraints. Other optimization approaches use for instance *stress minimization* [12] to a given stress function, often dependent on layout criteria such as proximity or topology, in order to generate the "best" non-overlapping state out of an initial layout. However, not all algorithms can be separated between these classes: There are also algorithms which try to combine or use advantages of multiple classes [11].

In this thesis, four major overlap removal algorithms are adopted: Two iterative constraint solvers, one being implemented in a well known physics engine, a greedy displacement algorithm for generating overlapping free and compelling visualizations and a cluster busting algorithm using a Voronoi technique.

These algorithms are compared with several metrics: Euclidean distance describes the displacement of a node from the initial to the adjusted layout. Area occupancy targets the compactness of the layout w.r.t. the used space and orthogonal ordering describes the relative positions of the nodes to each other, in order to generate an adjusted layout which resembles the initial layout structure as much as possible. These metrics are used to describe how good the new layout preserves the "mental map" [21] of the user. However, there is always a tradeoff between the compactness of an adjusted layout, with an packing algorithm on the one side, and the preservation of the structure of the layout, with an uniform scaling approach on the other side.

To conclude this thesis, a new layout algorithm has been designed which behaves better w.r.t. the given metrics, having a good tradeoff between compactness and structure. Measures will underlay these improvements.

2 Overlap Removal Methods

In this section, the four main algorithms for overlap removal are discussed: Two constraint solvers, one implemented in a well known physics engine, the tag cloud technique Wordle with its extension ManiWordle which uses a greedy layout approach and a Voronoi based technique which removes overlap by evening the object distribution.

2.1 Box2D Physics Engine

Box2D [1] is a widely used open source physics engine written in *C++* by Erin Catto. It uses an iterative constraint solver loop [6] for rigid body dynamics. A body is an entity in the physics engine, which has a shape and a position. Overlap is iteratively removed body-to-body wise by using a Baumgarte scheme [5] to push the bodies apart. The velocity of this push is proportional to the penetration depth of the two objects.

Figure 1: Overlap removal in Box2D: Solving body-to-body constraints by pushing bodies away from each other with applied forces.

The physics engine has the constraint that no bodies can overlap. By representing overlapping objects with bodies in the physics engine, the constraint solver removes existing overlaps.

Initially, an area in which the simulation can take place is created. This could for instance be the canvas rectangle. After adding all objects as bodies, the iterative constraint solver is started. The scaling down and up is necessary, because the physic engine has a very poor performance with very large numbers. Terms can easily have a size of 400×100, but in physic world terms, this might be the scale of a whole city. For better performance, everything is scaled down before being added to the physics engine. Afterwards, the results of the physics engine they are scaled up again. There is of course a tradeoff between performance and precision of the results.

The body of a term can be created in two different ways: a non-granular way in which the bounding boxes of the terms are used as their bodies, and an granular way in which the characters of a term are used to determine its body:

Algorithm 1: The Box2D overlap removal algorithm.

```
Data: List<Word, Position> input
Result: List<Word, Position> layouted
begin
    world = createPhysicsWorld()
    forall the word, position ∈ input do
        bounds ← getBounds(word, position)
        scaledBounds ← scaleDown(bounds)
        world.addToPhysicsWorld(scaledBounds)
    simulate()
    i ← 0
    List<Word, Position> layouted
    forall the body ∈ world do
        scaledBounds ← world.getFromPhysicsWorld(body)
        bounds ← scaleUp(scaledBounds)
        layouted.add(bounds,input.get(i).word)
        i ← i + 1
    return layouted
```

2.1.1 Non-granular Objects

The bounding box of the whole word is used to represent it in the physics engine. With this technique, the minimal distance between the two terms is limited by the bounding boxes.

Figure 2: Left: Using the bounding box of the whole term as its body. Right: Solved collision between two nongranular bodies.

2.1.2 Granular Objects

The bounding boxes of each character of a word are used to represent it in the physics engine. This reduces the minimal distance between the two terms in comparison to the non-granular technique.

Figure 3: Left: Using the bounding box of each character of a term to determine its body. Right: Collision between two granular bodies.

Words might get stuck in each other in the physics engine, if their characters lie in between each other. The constraint solver moves one body to the space between two characters of the second body. This results in an alternating movement of the body (here: left and right).

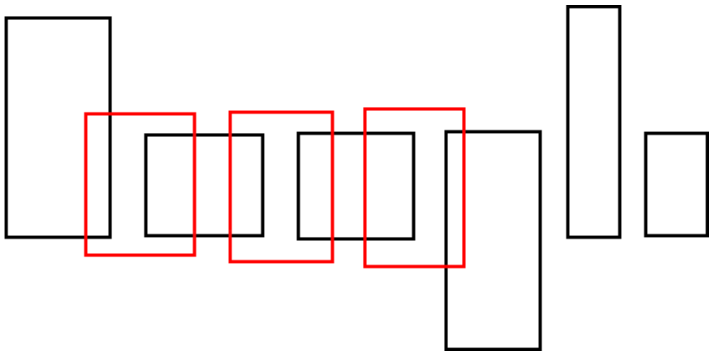


Figure 4: Problem situation for an iterative constraint solver: When trying to remove the overlap, the red term will be moved to the left in one iteration, leading to a still overlapping state, and to the right in the next iteration. A loop is created.

The solution is to make the bounding boxes of the characters bigger, so that they intersect the next/previous bounding box as can be seen in figure 4. With this technique, no other bounding box can get stuck in between. As a tradeoff for this correct overlap removing, space for layouting is lost, indicated yellow in figure 5.

Figure 5: Solution to the constraint solver loop problem: By filling the spaces between the characters no loop can occur.

2.2 Wordle (ManiWordle)

Wordle [4] is a web-based visualization tool to generate tag clouds [23] called "Wordles". It aims at generating aesthetic visualizations in terms of typography, color and composition.

For each term, a point is picked randomly, trying to keep the desired overall layout which can be specified in advance. If the term overlaps with any other already positioned term, it is moved in a spiral around its initial position.

Algorithm 2: The Wordle layout algorithm.

```
begin
    sort words by weight, decreasing
    forall the word w do
        w.position := makeInitialPosition(w)
        while w intersects other words do
            updatePosition()
```

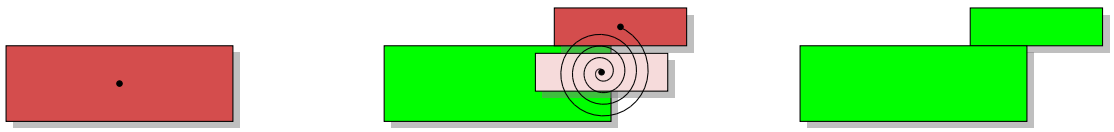


Figure 6: Wordle greedy layout algorithm: In case of overlap, search in a circular manner for a new position.

The only possibility to influence the result is to "play" with the parameters and try to achieve the wanted result this way. Koh et al. [16] try to overcome this weakness by providing more flexible control over Wordle by introducing ManiWordle, a Wordle based tool which allows the user to freely position terms. They also present some speed improvements over the original algorithm.

With this flexibility, the algorithm can be used to take the positions of overlapping terms as initial positions and with that, generate an overlap free layout that is close to the initial layout.

Algorithm 3: The Wordle (ManiWordle) overlap removal algorithm.

```
Data: List<Word, Position> input
Result: List<Word, Position> layouted
begin
    List<Word,Position> ORDLIST ← order input by weight, descending
    List<Word,Position> LAYOUTED
    forall the word, position ∈ ORDLIST do
        if !hasOverlap() then
            LAYOUTED.add(word, position)
            continue
        while hasOverlap() do
            newPosition = getNextPosition()
            if !hasOverlap() then
                LAYOUTED.add(word, newPosition)
                break
    return LAYOUTED
```

When a term cannot be positioned at the desired position due to overlap, a new position is generated in a circular manner around the initial position. One of the earlier described improvements of ManiWordle is, that the radius of the spiral depends on the height of the term. This accounts for the fact, that larger terms possibly need larger displacements to remove existing overlap.

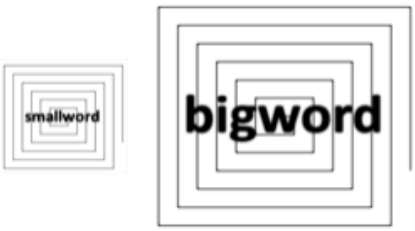


Figure 7: Adaptive spiral sizes for Wordle which account for the size of the term.

Another improvement is, that not every point on the spiral is used, but only points with a given distance from each other. This distance depends like the radius on the height of the term.

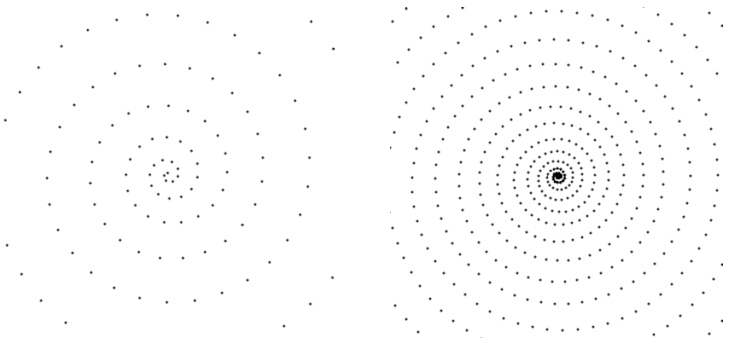


Figure 8: The distance between the sample points on the spiral adapts from large (left) to small (right) terms.

One of the reasons for the success of the Wordle layout is that words can not only be placed horizontally, but also vertically. If desired, the term can be rotated by 90° to the left or right. Other angles are not possible because having multiple terms with arbitrary angles (also upside down) might be hard to grasp. If a term does not fit at the given position, the algorithm tries to rotate it. If it fits, this position will be used, if not, the next position on the spiral is computed. An example for 90° rotation to the right is shown in figure 9.

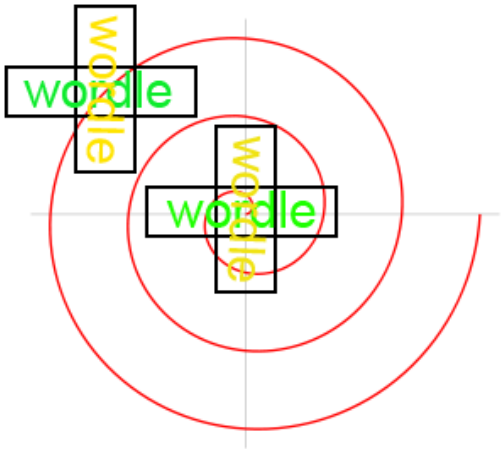


Figure 9: Rotation might be part of the search strategy.

The original Wordle has already partly been optimized in aspects of performance. The bottleneck is the test for intersections between words, because it is very expensive and if naively programmed, there are $(n+1) \cdot n$ intersection tests necessary where n denotes the number of already layouted words. A quadtree is used to minimize the number of these

tests and a "hierarchical bounding box" is created for each shape to avoid unnecessary intersection calculations.

In addition, ManiWordle focuses on multicore optimizations and intersection tests on images to speed up the algorithm. We will not focus on performance issues, because this thesis is a proof of concept, in which time is no important measure.

2.3 Voronoi

Llyod's algorithm [18], or Voronoi iteration, is an algorithm for relaxation of point distributions. It is well known in computer graphics because it can be used for position sampling for dithering. It starts with calculating the Voronoi region of every data point. The Voronoi region of a data point is the region in which no other generating point is closer. After this the centroid of these regions is determined by a given metric (usually the average of the Euclidean distance). Each generating point is then moved to its calculated centroid. These steps are repeated until the algorithm converges. After the convergence, every generating point of a Voronoi region is also its centroid: This state is called *centroidal Voronoi tessellation (CVT)*. The resulting distribution of the points is very even, that's why methods which create a CVT are also used to generate optimal meshes [8].

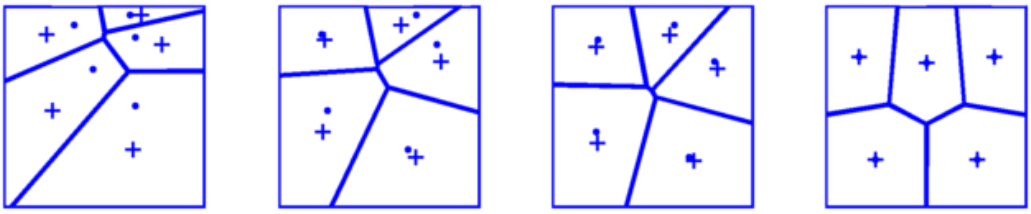


Figure 10: Images showing Lloyd's algorithm for an example point set from left to right: after the 1st, 2nd, 3rd and 15th iteration. Dots indicate the generating point, + the centroid of the Voronoi region. Images taken from [2].

In one dimension this algorithm has been shown to converge [9]. For higher dimensions, weaker convergence results have been shown [22].

The algorithm converges very slowly and suffers from problems like numerical precision. Therefore the algorithm is often said to converge when the result is "good enough" or the changes are below a certain threshold.

This algorithm is quite similar to our approach taken here. Our main goal is to use this technique to remove overlap from arbitrary shaped objects. The evening of the distribution is used to find a non-overlapping state.

But the arbitrary shaped objects pose a first problem in order to use this technique. Lloyd's algorithm only works on points. In order to determine the Voronoi regions of these arbitrary shapes 3D computer graphics can be used: We interpret a shape as a set of points and draw a cone through every point of the shape with the top of the cone at the points' position as shown in figure 11.

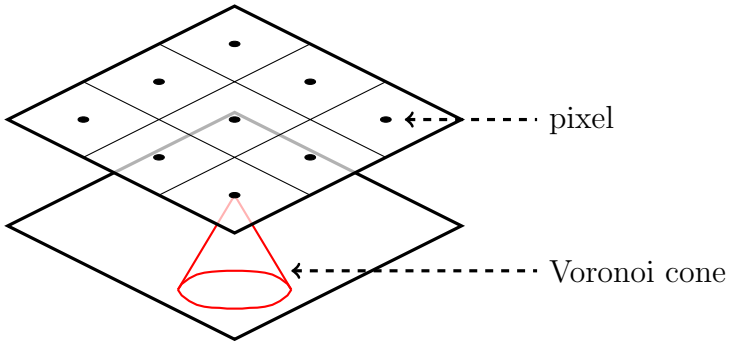


Figure 11: Using cones to determine Voronoi areas of pixels.

With this technique, every point which doesn't belong to a shape gets the color of the nearest cone (because of the structure of a cone). This is exactly how the Voronoi regions are defined. Calculating the centroid of the Voronoi region works the same way as described earlier in Lloyd's algorithm. But the translation vector is calculated in a different way: Instead of being described by one point, the new shape contains multiple points. The new translation vector is the vector from the initial center of all points of the shape to the new centroid of the corresponding region. All points of this shape get translated by this vector. The translations of every shape in every step are saved and summed up. After the algorithm converges, the summed up vector can be used to translate the original shapes once and remove their overlap. The convergence criterion must be determined by a certain threshold, because the translation vectors will never be zero due to numerical precision issues.

The shape of the objects are initially defined as vector graphics. As an initial step they have to been rasterized to a set of pixels (Voronoi points). These pixels are assigned an unique color which later helps to distinguish the different regions. After this, the Voronoi image is rendered in OpenGL with the cone technique described above. The translation of the Voronoi regions are calculated similar as in Lloyd's algorithm: the centroid of the pixels in the Voronoi image with the same color is computed. The translation vector described above is then applied to all generating points of the same color.

The outcome of the Voronoi image creation is highly dependent on the size of the cones. When they are too small, the Voronoi area around a point will only have a maximum distance depending on the scaling. The first approach was to use these "limited" Voronoi regions to guarantee compactness. Multiple problems occur when using this approach:

Algorithm 4: The Voronoi overlap removal algorithm.

```
Data: List<Word, Position> input
Result: List<Word, Position> layouted
begin
    initVoronoiPoints()
    image ← createVoronoiImage()
    while true do
        List<Vector> translations ← calcVoronoiMovement()
        i ← 0
        forall the vector in translations do
            translateVoronoiPoints(vector)
            input.get(i).translate(vector)
            i ← i + 1
        tmpImage ← createVoronoiImage()
        if changes in image > threshold then
            break
        else
            image = tmpImage
    return input
```

- Even if a Voronoi region has no overlap, it moves. This results from the uneven distribution of area in a word. Therefore, the center of the Voronoi region doesn't match the center of the object. Because this movement is clearly unwanted, every Voronoi regions "self vector", which is defined as the vector from the Voronoi region's center to the center of the object, is calculated initially to subtract this movement. For example the word "World"'s limited Voronoi region shown in figure 12 would move to the top, because more points are above its center than below.

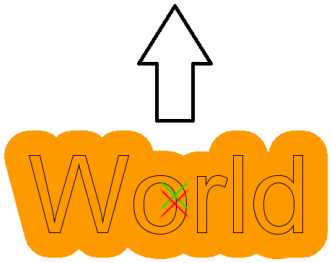


Figure 12: Limited Voronoi regions will move even without overlap because the center of the shape (denoted red) is not the same as the center of the Voronoi region (denoted green).

- Voronoi regions of smaller words or shapes can get stuck between or even inside (think of an 'O') other Voronoi regions. Overlap removal would fail in these cases. This can happen if the cone scaling is chosen too small.

These problems can be avoided by choosing the cone scaling just large enough. Voronoi regions have an infinite size in theory, only limited by other regions. Of course the range of the Voronoi regions must be limited to avoid the regions from drifting away. In practice, this is no problem because the clipping rectangle in which the Voronoi regions are calculated or drawn has a limited size. If a region moves out of the clipping volume, the "lost" area makes the region move back into the clipping volume, because the old center was closer to the center of the clipping volume than the newly calculated one.

A resulting new problem is, that this approach tries to divide the given space equally between every object. Therefore, the result will not be compact and the space between the objects will be approximately the same for each pair of words. This is clearly unwanted, because the compactness is an important part of the quality of the layout algorithm. To overcome this problem, the following technique is used:

- Calculate the area A occupied from the shapes of every object that should be arranged.
- Calculate an area $A^* = x \cdot A$. x is determined heuristically.
- The new rectangle R^* can have an arbitrary aspect ratio (here 1:1 because of compactness reasons), but must have the previously calculated area A^* and the side lengths $w = min(width(R^*), width(R))$ and $h = min(width(R^*), width(R))$, where R denotes the area which initially contained all the objects (canvas size). Move R^* as far as possible to the center of all shapes, without leaving R.
- If objects lie outside this rectangle, scale every position w.r.t. the new center, so that the rectangle contains every center.
- Calculate the Voronoi movement in this smaller Voronoi image.

This leads to a) more compact and b) parameterizable results, which can compete with the other two algorithms in terms of compactness.

Because in every iteration of this algorithm an image is created and the algorithm might possibly need many iterations to converge, performance is an important issue. To speed up the image creation, OpenGL is used as renderer. Drawing cones through every pixel of a shape is very costly. So instead, there are 2 classes of points which belong to the shape. The first class of points are the points that belong to the *outline* of the shape. At these pixels cones have to be drawn, because they are closest to any yet uncolored points outside

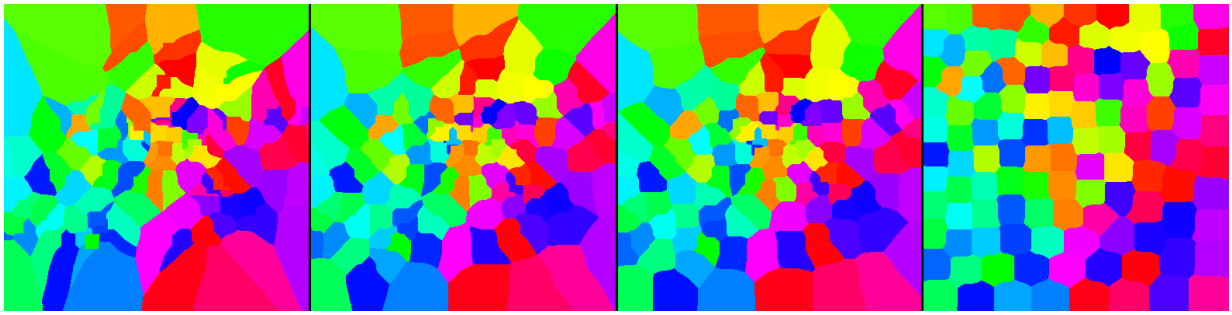


Figure 13: The Voronoi regions of overlapping rectangles in (left to right) the 1st, 2nd, 3rd and 50th iteration.

of the object. The points *inside* the shape don't have to be drawn as cones, because the cone (outline) points are always closer to any point outside the shape. They can simply be drawn as dots to speed up the image creation process significantly.

2.4 VPSC

This algorithm was designed by Dwyer et al. [10] for removing node overlaps in graph layouts. But this is no restriction for our purposes, because objects, with initial positions, can easily be extended to a graph by using the objects as nodes and adding every possible edge, creating a complete graph. This algorithm consists of two parts: The first part generates separation constraints for nodes. The second part tries to find a solution to all of these constraints as near as possible to the initial node placements.

In the separation generation step, a sweep line along an axis is used to generate non overlapping constraints in one dimension: When a node is found by the sweep line, its neighbors in the orthogonal direction to the scanline direction are computed and two non overlapping constraints (between each of the neighbors and the found node) are generated. The neighbor finding process that decides with which neighbor the node should have non overlapping constraints is an heuristic one. Indexing structures like red-black trees are used to speed up the neighbor finding process. Also, redundant or transitive constraints can be removed to keep the number of constraints low.

After generating constraints of the form $u + d \leq v$, where d is the minimal gap between the two nodes and u, v are the positions of two nodes, the following constrained optimization problem has to be solved for each dimension, described by Dwyer et al. [10]:

> Variable placement with separation constraints (VPSC): Given n variables $v_1, \ldots, v_n, a$ weight $w_i \geq 0$ and a desired value d_i, for each variable and a set of

separation constraints C over these variables find an assignment to the variables which minimizes $\sum_{i=1}^{n} w_i \times (v_i - d_i)^2$ subject to C.

The set of separation constraints is then treated as a weighted directed graph called *constraint graph* with the nodes from the former graph and the separation constraints between nodes as the edges. The variables (positions) are then processed in ascending order. Before processing vertex v_i, all previous vertices $v_1, \ldots, v_{n-1}$ already have to satisfy the separation constraints. The already positioned vertices create an already solved "block". Before adding v_i to the block, the separation constraints between v_i and the block have to be solved. With this technique, the algorithm merges nodes into larger and larger blocks until all nodes satisfy the separation constraints.

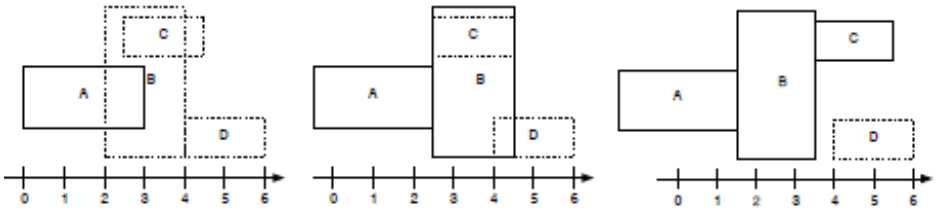


Figure 14: Example for an optimal VPSC solution taken from [10].

Figure 14 shows the layout process for four example nodes with rectangular shapes. The variables (positions) are processed in ascending order, so the boxes are positioned in the order A,B,C,D. Box A is positioned at its desired position (a), because no other box is considered yet. It is added to an empty block. The next box to layout is B, which overlaps with the already positioned block containing A. The separation constraints between box B and the block are solved by moving them away from each other (b). With no overlap remaining, box B is added to the block containing A. Next, box C is considered which overlaps with the block containing A and B. After solving this constraint, box C can be added to the block AB (c). Box D does not overlap with the already positioned block now containing ABC, so it can simply be placed at the desired position and added to the block.

There might however be non-optimal solutions depending for example on the ordering of the nodes. A solution is considered non-optimal, if there is another solution which performs better to given metrics, mainly the euclidean distance between desired and final position. Improving the solutions can be achieved by splitting blocks into smaller blocks.

3 Evaluation

Shapes in the layout algorithm can be described by points, most common by their centers. A layout adjustment algorithm then transforms a set of Points denoted by P with initial position p and coordinates x_i and y_i to a position p' with coordinates x'_i and y'_i accordingly. The resulting layout quality highly depends on the differences between the initial and the resulting layout. Misue et al. [21] use the term "mental map" to describe the image a user has of an initial layout. If, for example, the layout adjustment algorithm completely rearranges the layout, the user's mental map of the layout is destroyed. They measure the quality of a new layout in terms of how good the mental map is preserved. To measure this preservation, the quality of the resulting layouts is measured with several metrics analogous to the mathematical models described by Misue et al.: *Euclidean Distance* attempts to capture the idea that an object should be close to its original position after the adjustment. *Area Occupancy* describes how compact the layout is. As a last measures, the *Orthogonal Ordering* is used to describe that the relative positions of the nodes to each other should be preserved.

3.1 Methods

3.1.1 Euclidean Distance Measure

The Euclidean Distance measure is a simple metric which measures the average distance moved by each Point from p to p'. These points represent the center of the shapes they belong to.

This measures represents the idea that if points move far away from their initial position the second layout will look very different.

$$M_{euclidean} = \frac{1}{|P|} \sum_{1 \leq i \leq |P|} d(p_i, p'_i) \tag{1}$$

Where $|P|$ denotes the number of points and $d(p_i, p'_i)$ the Euclidean distance between the position p and the new assigned position p'. The Euclidean distance is defined as

$$d(p, p') = \sqrt{(x' - x)^2 + (y' - y)^2} \tag{2}$$

3.1.2 Area Occupancy

Instead of working on points representing a shape, this method works on the surrounding rectangles r of a shape. R is the set that contains the rectangles of all shapes. It calculates the rectangle r_{all} which is the minimal rectangle that contains all other rectangles $r \in R$. Returned is the fraction of used space in r_{all}.

$$M_{occupancy} = \frac{\sum_{r \in R} r}{r_{all}} \tag{3}$$

Where r denotes the area of the rectangles.

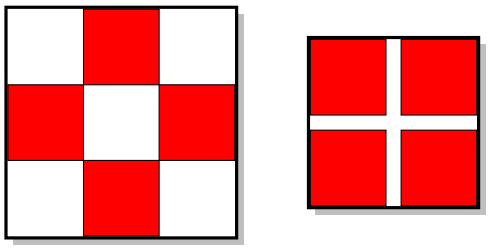


Figure 15: The surrounding rectangles r filled red and the complete surrounding rectangle r_{all} around them show, that the *Area Occupancy* metric describes how "well" space is used by the rectangles.

3.1.3 Orthogonal Ordering

The *Orthogonal Ordering* metric describes changes in the relative positioning of the objects. This measure works on the ordering between two old positions p_i, p_j and the corresponding new positions p'_i, p'_j.

In two scan line sweeps, one horizontally and one vertically, the orderings of all points before and after the layout adjustment are saved in lists L_x, L_y and L'_x, L'_y respectively. The change in the orthogonal ordering is measured as the number of inversions between the vertical and horizontal lists.

$$M_{inversions} = \sum_{d \in x,y} \sum_{i<j} inv_d^{(t)}(i,j) \tag{4}$$

$$inv_x^{(t)}(i,j) = \begin{cases} 1, & \text{if } sgn(x_i^{(t)} - x_j^{(t)}) \neq sgn(x_i^{(t-1)} - x_j^{(t-1)}) \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

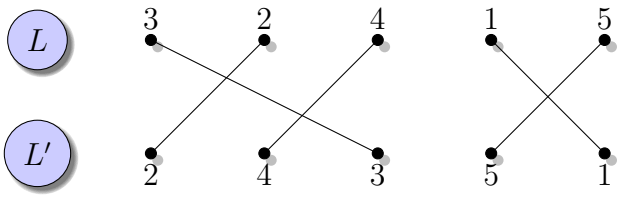


Figure 16: Each crossing line pair represents a pair of numbers, that changed their positions between the two lists L and L'. The number of intersections equals the number of inversions.

Where i, j are indices of list L and $inv_y^{(t)}(i, j)$ is defined analogous to $inv_x^{(t)}(i, j)$

4 Results

As the metrics have already been discussed in previous section, our focus now lies on the test data sets. In order to generalize the results, abstract shapes (rectangles) are used in these data sets. For comparison reasons, the Wordle layouting strategy will not contain rotation.

There are three different data sets, which are used with three different kind of object shapes:

- **Square**: The object's shape is a square (1:1 aspect ratio).
- **W** > **H**: The width of the objects is bigger than its height (3:1 aspect ratio).
- **H** > **W**: The height of the object's shape is bigger than its width (1:3 aspect ratio).

There are no mixtures of objects with different shapes in a data set, in order to detect possible special cases in which the behavior of an algorithm depends on the shapes of the objects. The detailed measuring results are at the end of every data sets section plotted in bar charts for every metric.

4.1 Dataset: Separated Cluster

The first data set contains two well separated clusters. The points in the clusters are assigned randomly in the area that is occupied by the cluster, described by a rectangle. To distinguish the objects belonging to each cluster after the layout adjustment, they are colored differently.

Results:

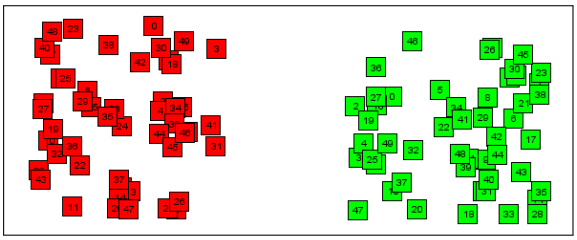


Figure 17: Initial

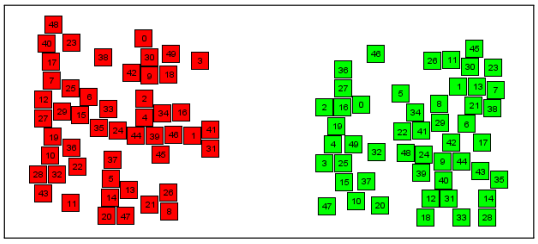


Figure 18: Box2D

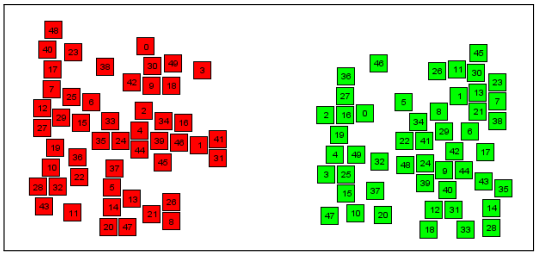


Figure 19: VPSC

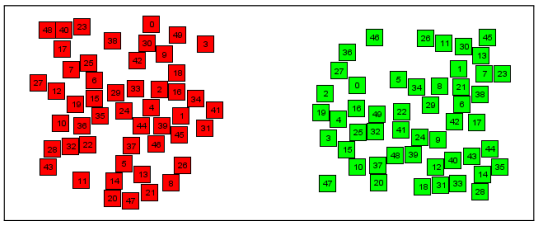


Figure 20: Wordle

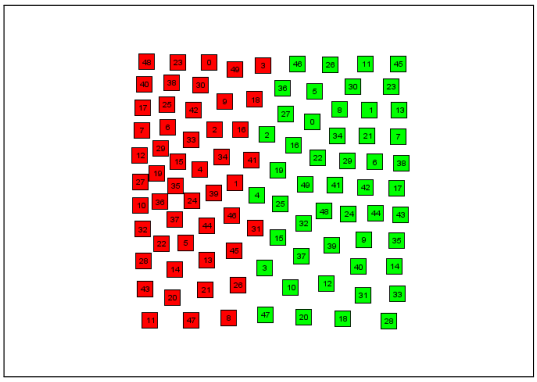
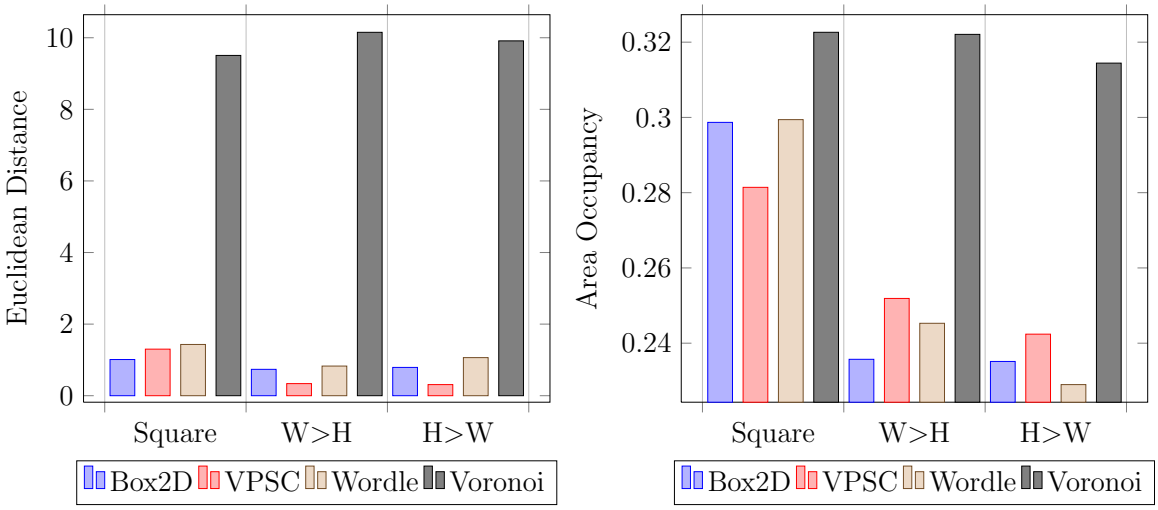
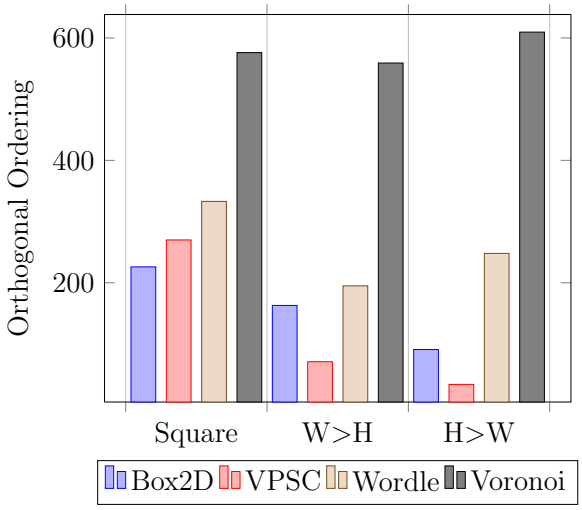


Figure 21: Voronoi





In the Euclidean distance metric, Box2D, VPSC, and Wordle perform similar. Box2D performs better for square objects, VPSC performs better for the other two shapes. Voronoi overall performs worse than the other three algorithms.

In the area occupancy metric, Box2D performs good for square objects, but VPSC better for the other two shapes. Voronoi's good performance in this measure is mainly due to its density optimization and is therefore not very meaningful.

In the orthogonal ordering metric, Box2D again performs best for square objects, VPSC for the other two shapes. Wordle performs worse with each shapes, Voronoi even worse than Wordle.

4.2 Dataset: Overlapping Cluster

The second data set contains 3 overlapping clusters, which are less dense than in data set 1. The points in the cluster are again randomly distributed in 3 overlapping rectangles.

Results:

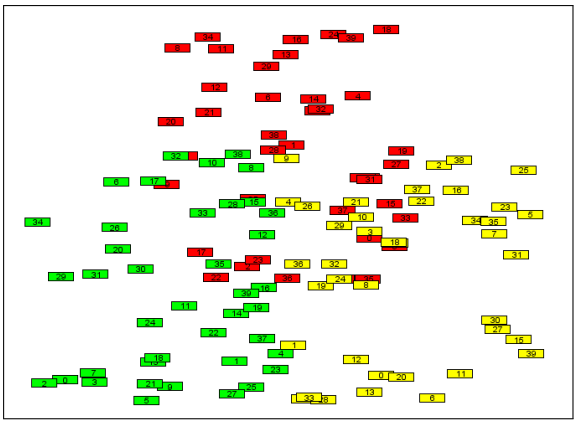


Figure 22: Initial

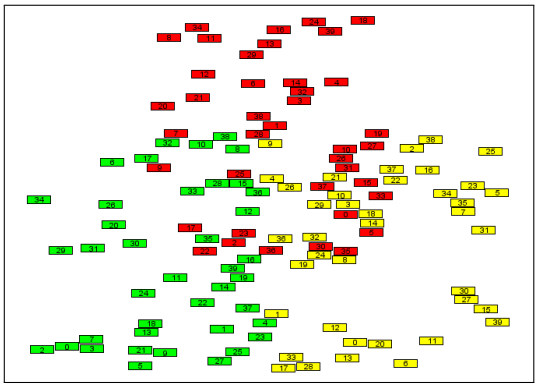


Figure 23: Box2D

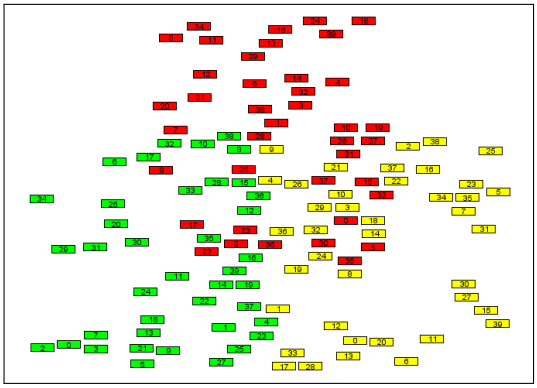


Figure 24: VPSC

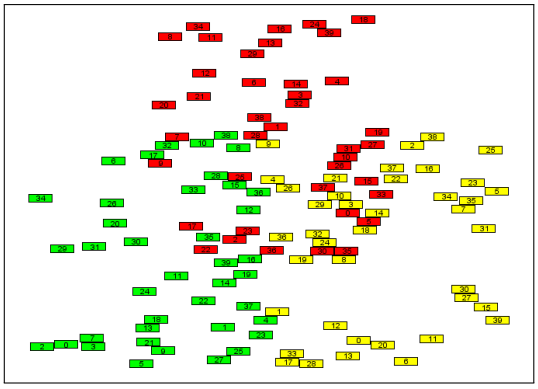


Figure 25: Wordle

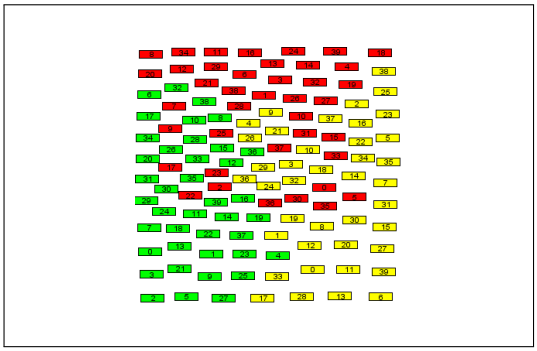
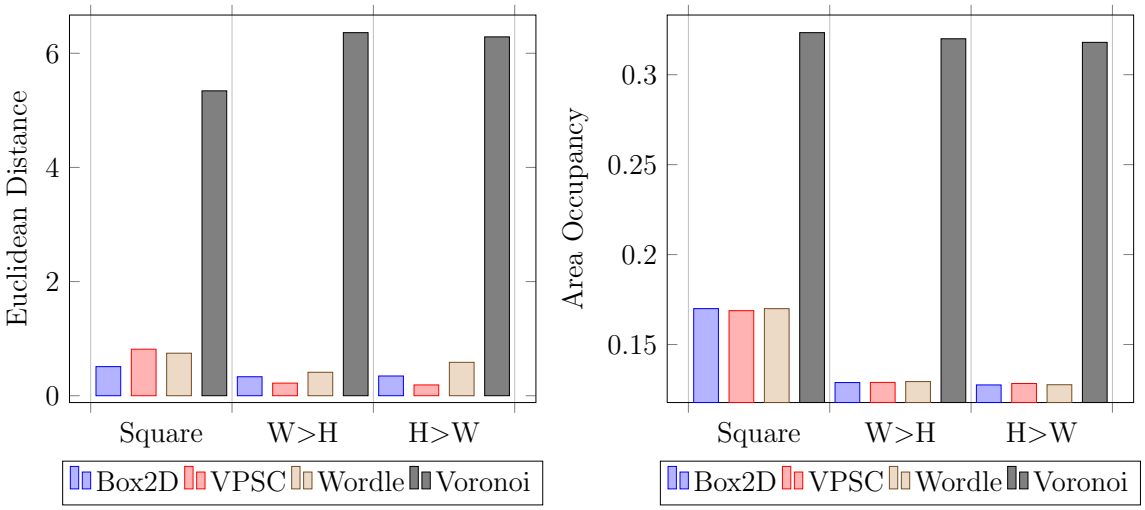
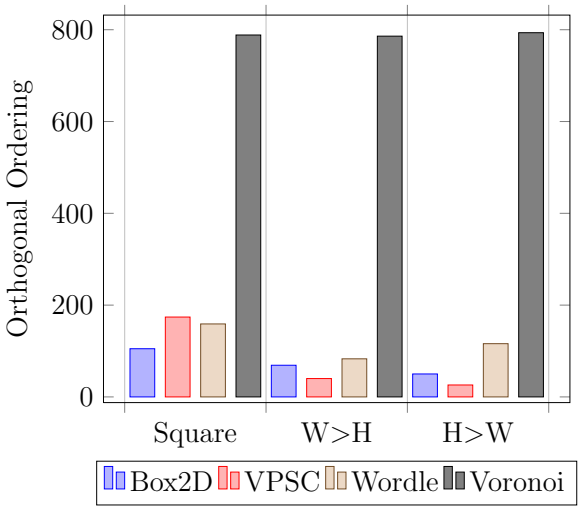


Figure 26: Voronoi





In the Euclidean distance metric, Box2D shows a similar performance like in the previous

dataset: It performs best for squares, but VPSC performs better for the other two shapes. Wordle performs slightly worse and Voronoi's results are far off.

In the area occupancy metric, Box2D, VPSC, and Wordle perform almost identical, whereas Voronoi performs worst again. The close results for the other three algorithms are due to the little overlap and the sparse data set, which allows to remove overlaps body-to-body wise without having to deal with additional overlap from these changes.

In the orthogonal ordering metric, the results are quite similar to the results in the Euclidean distance metric: Box2D performs best for squared shapes, VPSC for the other two. Voronoi's results are worst here too.

Overall this dataset has similar results to the previous one.

4.3 Dataset: Uniform Distribution

The last data set contains one large, dense, evenly distributed cluster. The points are randomly distributed, this time in a single rectangle.

Results:

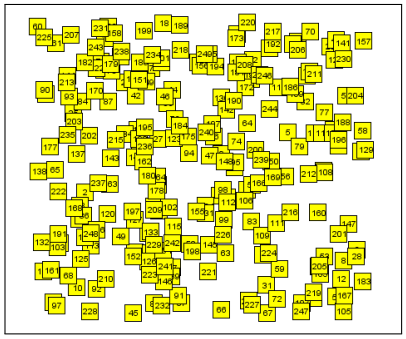


Figure 27: Initial

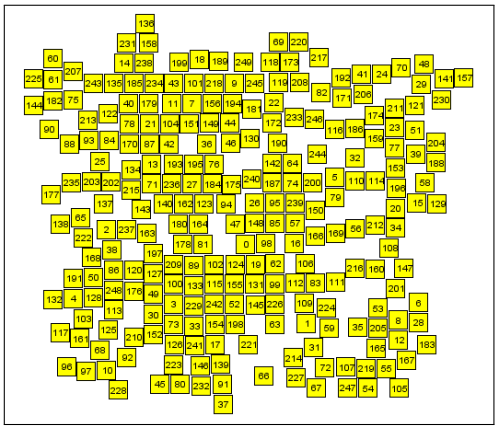


Figure 28: Box2D

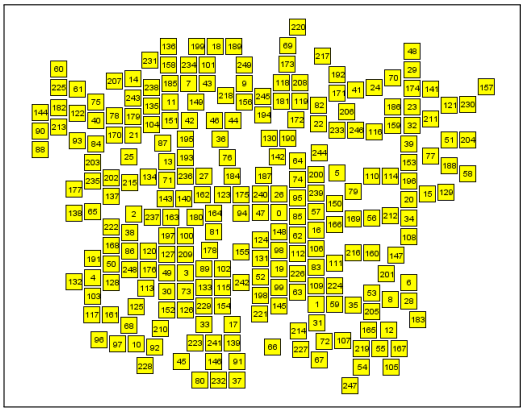


Figure 29: VPSC

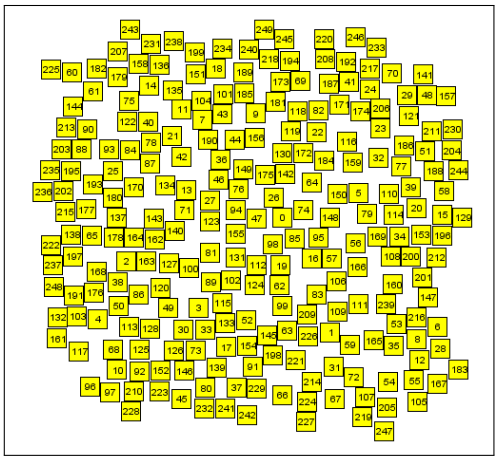


Figure 30: Wordle

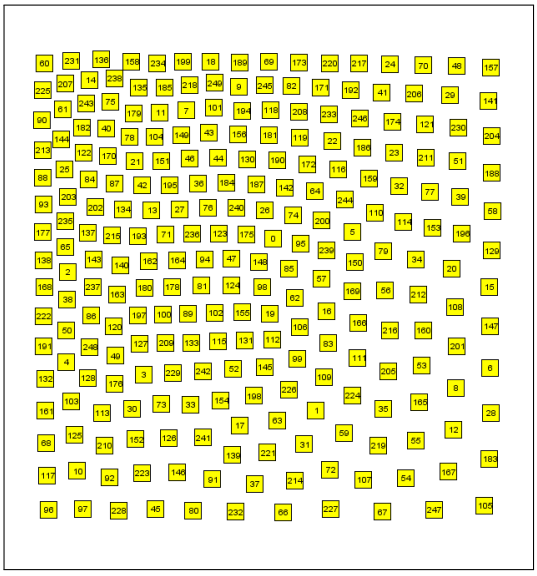
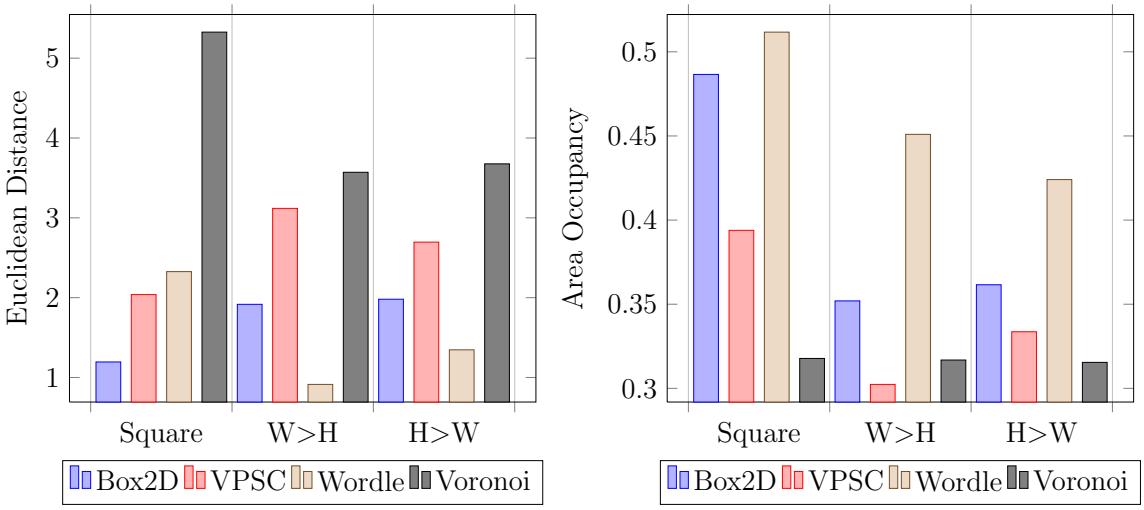
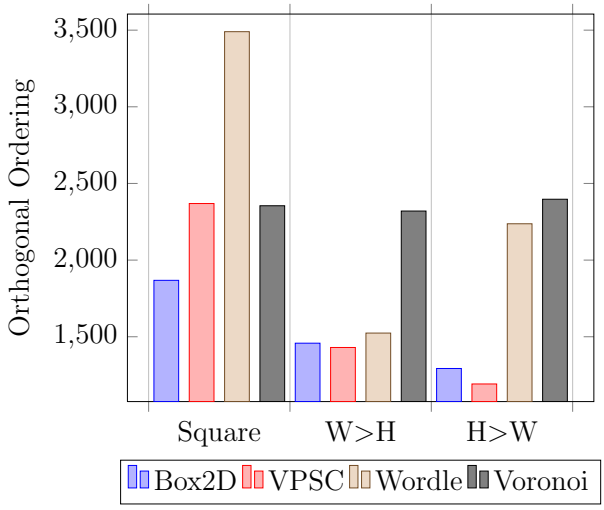


Figure 31: Voronoi





In the Euclidean distance metric, Box2D performs best for squared shapes, but Wordle performs better for the other two shapes. VPSC does not perform as well as in the previous two datasets, it almost performs as bad as Voronoi.

In the area occupancy metric, Wordle outperforms the other algorithms with each shape. Box2D performs second best, VPSC performs weak with the W>H shape. Voronoi performs very similar for each shape.

In the orthogonal ordering metric, Box2D performs best again for squared shapes, but VPSC performs better for the other two shapes. Wordle does not perform very good, especially with the squared shape and Voronoi performs again similar bad for each of the shapes.

4.4 Intermediate Conclusion

Box2D outperforms the other algorithms in the first two datasets with squared shapes, VPSC with the other two shapes. The two algorithms overall seem to have a similar behavior. The third dataset shows a different behavior: Box2D and VPSC outperform the other algorithms w.r.t. orthogonal ordering, but Wordle has a much better compactness in comparison with Box2D which is expressed by a lower Euclidean distance and higher area occupancy. Wordle lacks in terms of orthogonal ordering, because the search strategy is basically a circle, which destroys any initial ordering if the displacement of the circle gets too large. The main difference between the behavior of the algorithms in the first and second dataset versus the third dataset is the density of the objects and with this the number of overlaps. Box2D and VPSC have similar issues with the third dataset in terms of compactness.

5 Improvements

Knowing the strengths and the weaknesses of the different algorithms now, the following sections aim at creating an improved or new overlap removal algorithm, which should perform better than the given algorithms.

Box2D has an issue with overlap removal when objects with similar aspect ratio overlap and lie in a row. Because a force in the same direction will be applied on every object, the resulting layout will have "stacked" objects and the overall layout will not be compact.

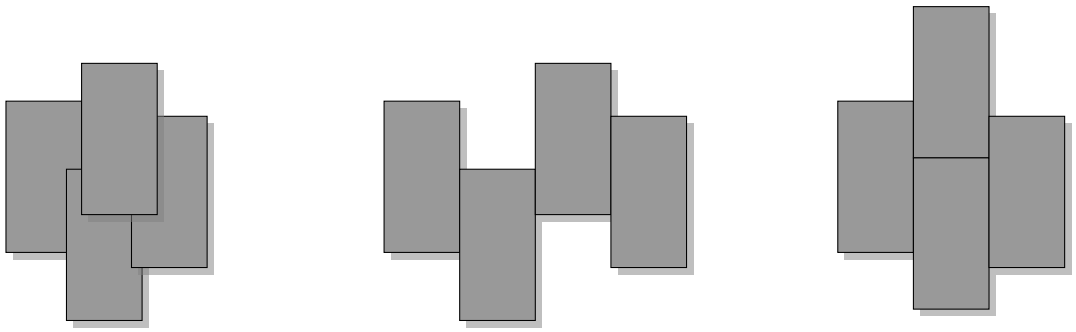


Figure 32: Overlap removal in the same direction may lead to a stacking problem (center) even if an other, more compact layout would be preferable (right).

A similar problem exists for VPSC, because the separation constraints and the displacement are done axis-wise.

5.1 Scanline Wordle

5.1.1 Algorithm

The idea of the new algorithm is to improve the orthogonal ordering of Wordle. The new algorithm uses a modified version of Wordle's greedy layout strategy: It adopts the scanline overlap removal strategy of the *Force Scan* [21] algorithm, hence the name *Scanline Wordle*.

As a first step, all objects are ordered by a sweep line scan along one axis. Which axis depends on the distribution on the positions. In order to keep the overall structure of the initial layout, the sweep should be on the axis with a larger range of positions. After this, the objects are layouted in a greedy manner. If there is no overlap, the object is layouted at this position. If there is overlap, Wordle is used to calculate a new position. When

generating the layout in a structured manner (in contrast to the original Wordle layout strategy which only works with the weight of the elements), the hope is to minimize the displacement and thereby improve the overall orthogonal ordering.

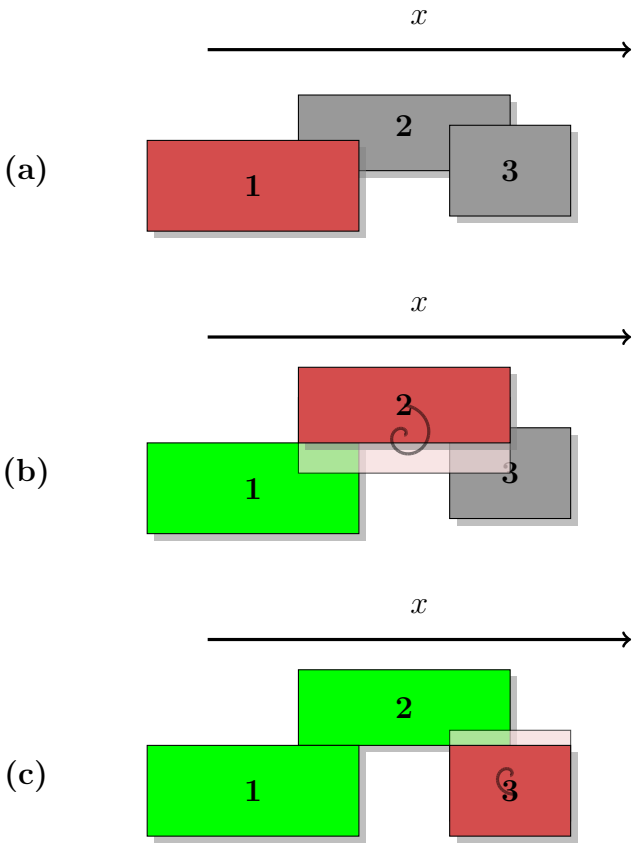


Figure 33: The greedy layout strategy in Scanline Wordle: Elements are layouted from left to right in each step. In case of no overlap (a), the element is layouted at this position. If overlap occurs, a new position is found via the Wordle displacement strategy.

Because of the order the objects are processed (here: low x-coordinates first), the layout will extend to one side (high x-coordinates). This is clearly unwanted, because in order to preserve the mental map of the user, the adjusted layout should have the same central location as the initial layout. To overcome this issue, the initial center x-coordinate x is saved. After the layout adjustments, every object is moved along a vector $x' - x$, where x' denotes the center in the adjusted layout.

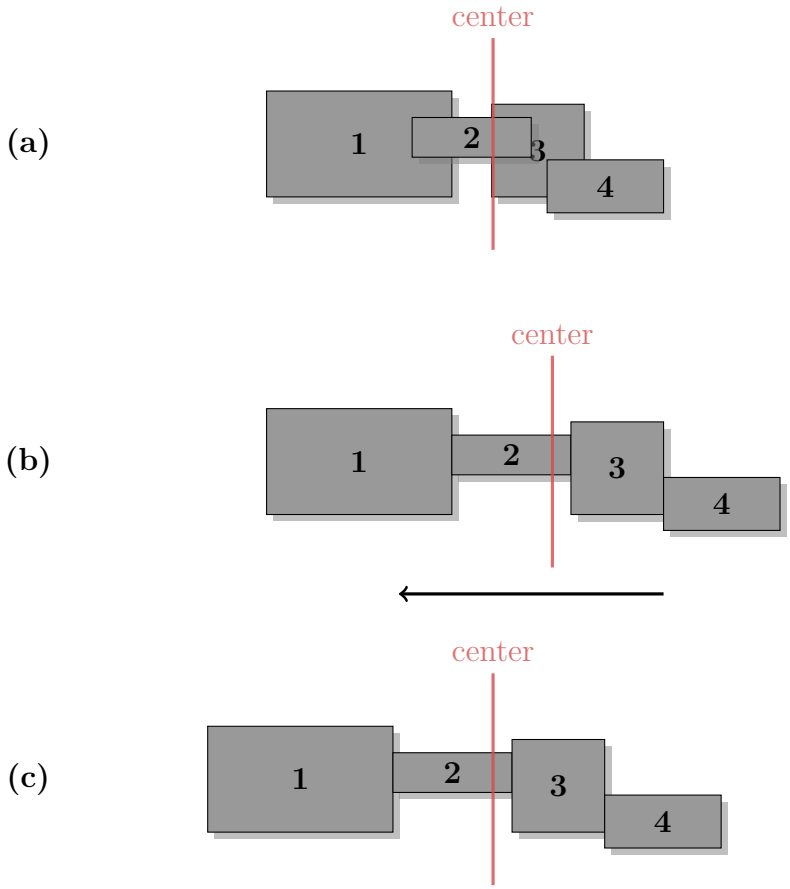
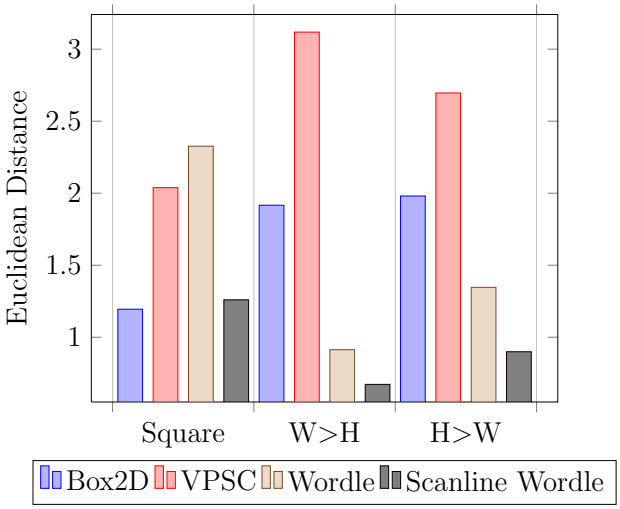
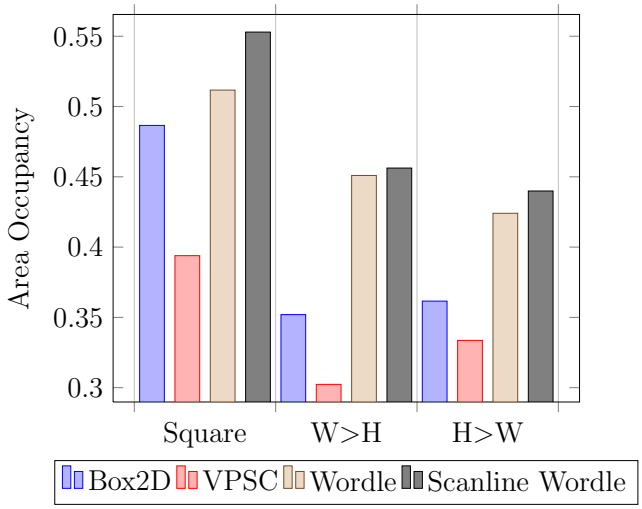


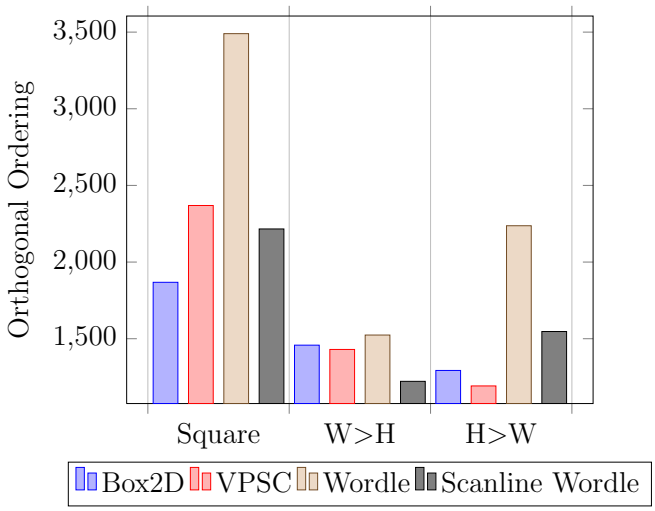
Figure 34: The adjusted layout is transformed in order to keep the central layout position.

5.1.2 Results

For this measure, the "Uniform Distribution" dataset is used.

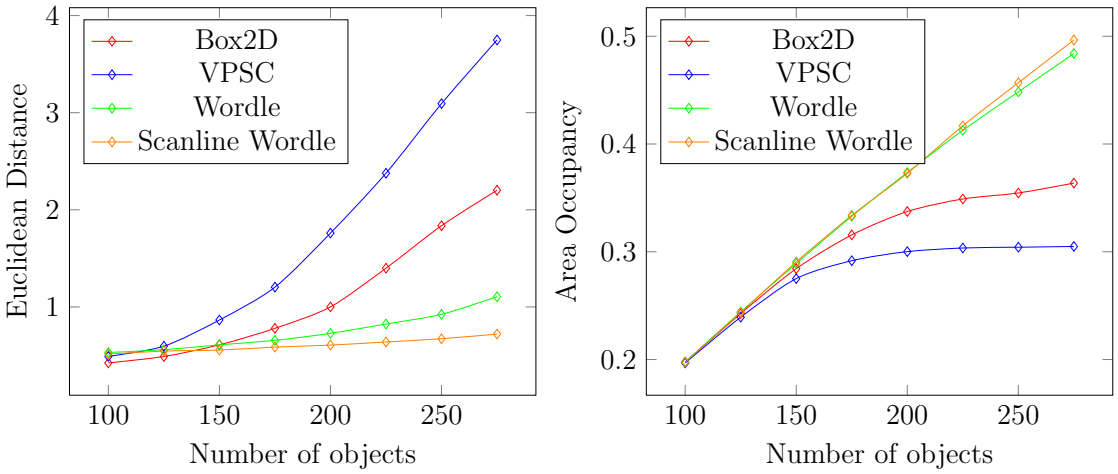


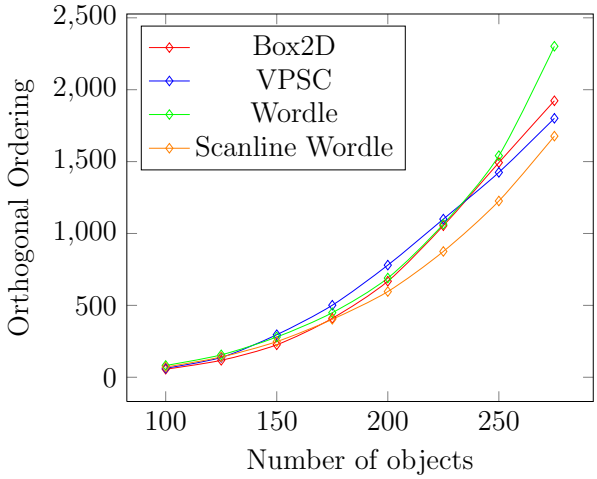




Scanline Wordle performs better than Wordle, Box2D and VPSC in the Euclidean distance and the area occupancy. In the orthogonal ordering metric, it performs about as good as Box2D and VPSC. The case W > H is of special interest, because the algorithm will mainly be used for word clouds and these shapes resemble words the most. In this case, Scanline Wordle outperforms the other two algorithms. Because the first and the second sparse datasets performed very similar, the density seems to be an important factor for the layout algorithms.

In order to identify how each of the three algorithms, Box2D Wordle and Scanline Wordle, behaves for different densities, the density of the data set 3 is changed and the quality of each algorithm is measured. The density is changed by modifying the number of random positioned objects in the given area.





The diagrams show, that Scanline Wordle scales better than the other two algorithms with increasing density. The strong increase in Euclidean distance and the slow increase in area occupancy beyond a certain density are due to the stacking problem.

5.1.3 Scanline Wordle & Real World Data

Yet all measures have been done on abstract shapes like rectangles or squares. The real purpose of this layout adjustment algorithm however, is to remove more complex overlaps with text. The following examples use "real world" examples and data to show that Scanline Wordle does not only work well on abstract data, but also on "real" data.

Map of Germany

The following example contains a map of the biggest cities in Germany. With small labels for every node (city), there is almost no overlap. When increasing the size of the city labels however, overlap occurs more often. Here, the size of the labels has a minimal font size and increases with the number of inhabitants. The displacement of every tag is visualized in the image next to it: The more red the tag is drawn, the higher its displacement in Euclidean distance.

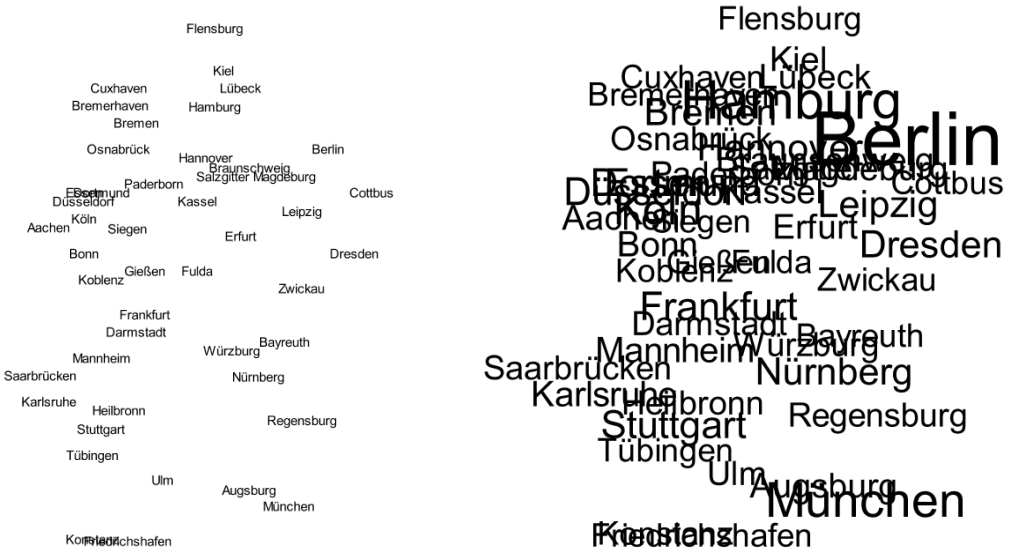


Figure 35: Left: A map of Germany containing the largest cities and their locations. Right: The same map containing bigger labels with a minimal size which is linearly increasing with the number of inhabitants.

Now, we use the four (Box2D, VPSC, Wordle, Scanline Wordle) layout algorithms to remove this overlap.

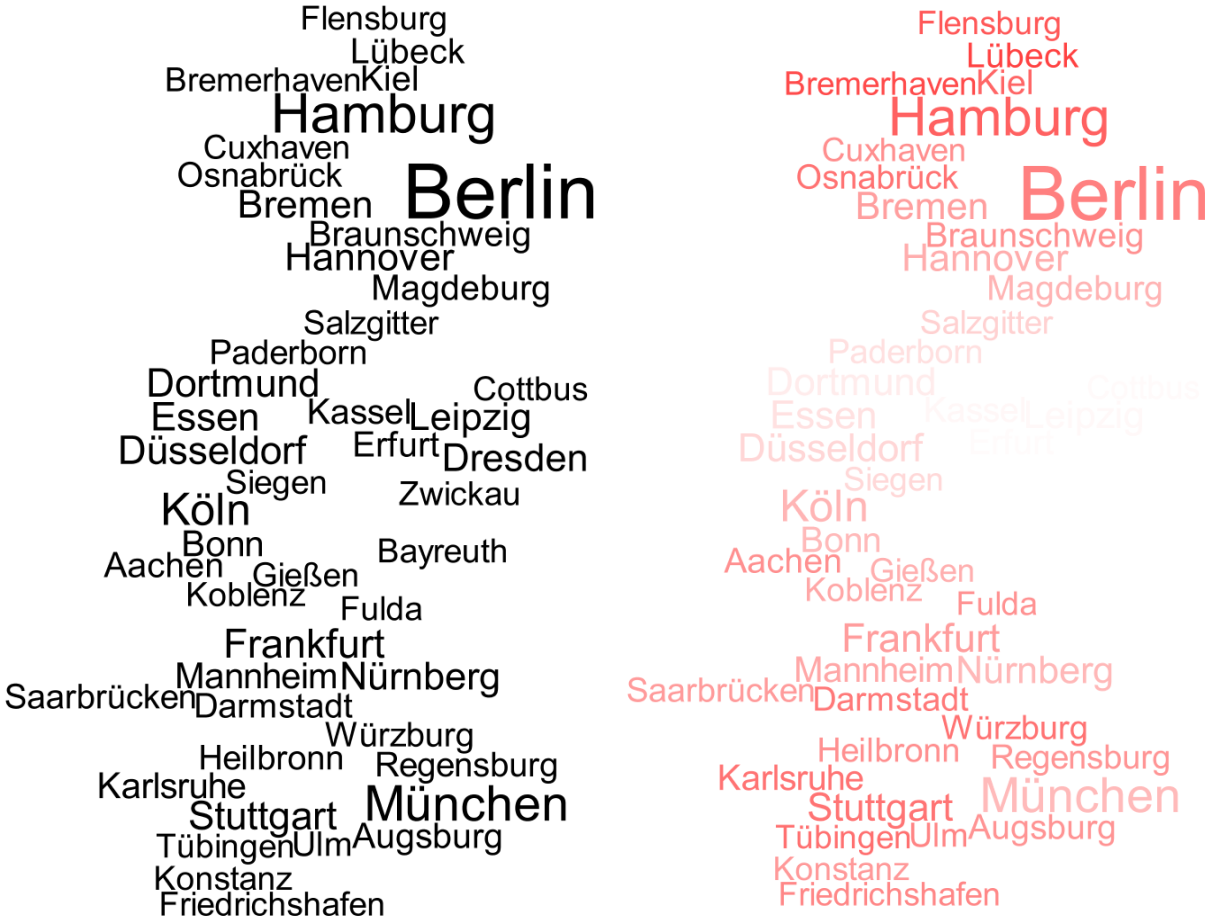


Figure 36: Left: The map of Germany with overlaps removed by *Box2D*. Right: Amount of displacement (Euclidean distance) of the tags indicated red.

Figure 37: Left: Map of Germany with overlaps removed by *VPSC*. Right: Amount of displacement (Euclidean distance) of the tags indicated red.

Figure 38: Left: Map of Germany with overlaps removed by *Wordle*. The layout is compact, but lacks the structure of the initial layout. Right: Amount of displacement (Euclidean distance) of the tags indicated red.

Figure 39: Left: Map of Germany with overlaps removed by *Scanline Wordle*. The layout is compact and the initial structure is preserved. Right: Amount of displacement (Euclidean distance) of the tags indicated red.

	Euclidean Distance	Area Occupancy	Orthogonal Ordering
Box2D	20.71	0.37	88
VPSC	25.92	0.35	**32**
Wordle	12.61	0.49	230
Scanline Wordle	**9.47**	**0.52**	146

Table 1: The measures of the Germany dataset in detail.

Box2D and VPSC fail to convey the informations of the map due to the stacking problem. Their results are stretched too much in one direction. Wordle has a bad orthogonal ordering score, which distorts the overall city positions very much. Scanline Wordle has a good tradeoff between orthogonal ordering and keeping the structure of the initial layout. The measures can be seen in detail in table 1.

Map of England

The following example contains a map of the biggest cities in England. With small labels for every node (city), there is almost no overlap. When increasing the size of the city labels however, overlap occurs more often. Here, the size of the labels has a minimal font size and increases with the number of inhabitants. The displacement of every tag is visualized again in the image next to it: The more red the tag is drawn, the higher its displacement in Euclidean distance.

Figure 40: Left: A map of England containing the largest cities and their locations. Right: The same map containing bigger labels with a minimal size which is linearly increasing with the number of inhabitants.

Again, we use the four (Box2D, VPSC, Wordle, Scanline Wordle) layout algorithms to remove this overlap.

Figure 41: Left: The map of England with overlaps removed by *Box2D*. Right: Amount of displacement (Euclidean distance) of the tags indicated red.

Figure 42: Left: Map of England with overlaps removed by *VPSC*. Right: Amount of displacement (Euclidean distance) of the tags indicated red.

Figure 43: Left: Map of England with overlaps removed by *Wordle*. The layout is compact, but lacks the structure of the initial layout. Right: Amount of displacement (Euclidean distance) of the tags indicated red.

Figure 44: Left: Map of England with overlaps removed by *Scanline Wordle*. The layout is compact and the initial structure is preserved. Right: Amount of displacement (Euclidean distance) of the tags indicated red.

As in the Germany dataset, Box2D and VPSC fails to convey the informations of the map due to the stacking problem. Their results are stretched in a similar fashion. Wordle again has a bad orthogonal ordering score, which distorts the overall city positions. Scanline Wordle has a good tradeoff between orthogonal ordering and keeping the structure of the initial layout in this dataset too. The measures can be seen in detail in table 2.

	Euclidean Distance	Area Occupancy	Orthogonal Ordering
Box2D	22.17	0.28	62
VPSC	20.57	0.28	**31**
Wordle	11.37	**0.39**	189
Scanline Wordle	**8.84**	**0.39**	120

Table 2: The measures of the England dataset in detail.

Semantical Zoom Data

A previous project dealt with the scaling of textual documents. As a part of this project, a semantical zoom technique was developed, which allowed to view documents at any scale, while preserving keywords in order to skim the document. The technique kept the structure of the document (sections) and scaled the non-important text, while leaving the most important keywords the same size. When trying to write these keywords at their scaled positions, the key terms would overlap due to the "growing" size in comparison to non keywords. For overlap removal, Box2D was used, which has the already known stacking problem.

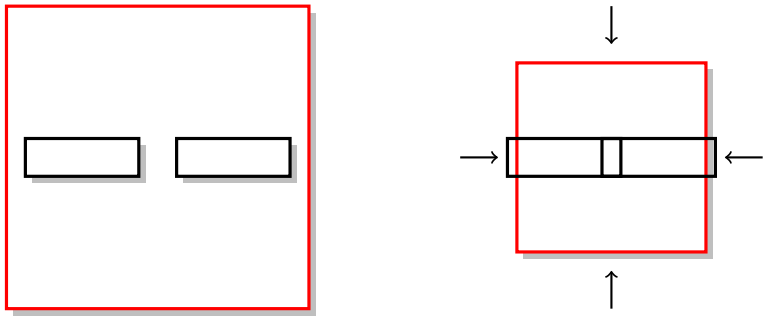


Figure 45: When scaling a section (red), Box2D might not find a suitable solution with no overlap for the key terms (black) due to the stacking problem.

When replacing the overlap removal algorithm with Scanline Wordle, this problem will not occur.

Also, the minimal scaling size was previously limited by Box2D, because the stacking problem would occur very often under a certain scaling. With Scanline Wordle, the new limiting size of the lower scaling is the width of the term.

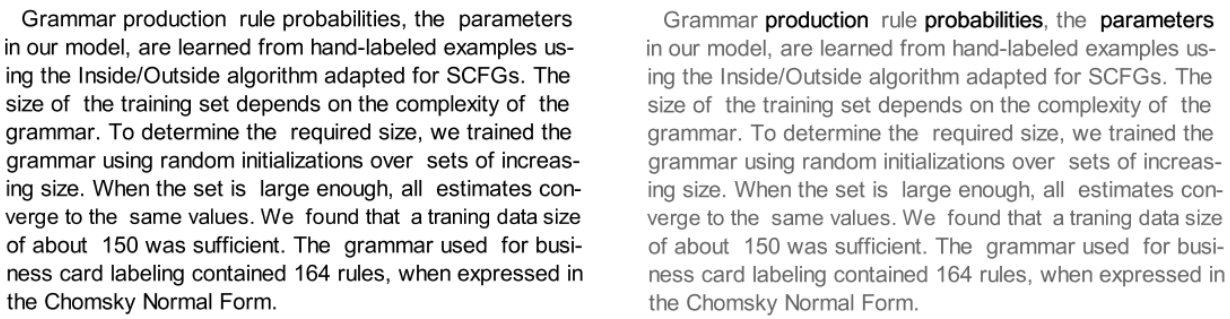


Figure 46: Left: Section of a text document before semantic scaling. Right: Keywords of this section highlighted . Excerpt from [13].

Figure 47: Text section after semantic scaling. Box2D fails due to the stacking problem, whereas Scanline Wordle succeeds.

Figure 48: Using *Scanline Wordle*, the new lowest scaling size is the maximal width of the terms.

MDS Data

For this data set, the top news from known news pages (CNN, BBC) were fetched over the time of a day. Then, the full text of all articles is processed with OpenCalais [3] to extract entities (in this case, persons). After this, the occurrence of each pair of entities is counted in each article. The two entities with the most common occurrences have the distance 0, entities which do not occur together have the distance 1. Entity pairs with occurrences between the maximal number of occurrences and 0 are linearly scaled in between. This creates a complete graph, where the edges denote the distance between the points. Because of the distance scaling between 0 and 1, the data set is scaled up for displaying them. The shown data was fetched on 1. March 2011.

The terms belonging to the same cluster have the same color.

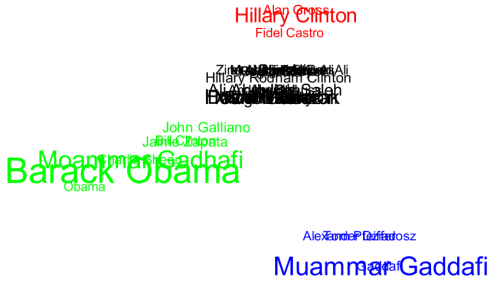


Figure 49: Initial

Scanline Wordle and VPSC are the only algorithms which can keep the red cluster together. VPSC has a very good orthogonal ordering, but stretches the layout very much. That's why Scanline Wordle performs better in terms of Euclidean distance and area occupancy. Overall, the Scanline Wordle layout has a better tradeoff between keeping the structure and keeping the layout compact. The measures can be seen in detail in table 3.

	Euclidean Distance	Area Occupancy	Orthogonal Ordering
Box2D	25.8	0.22	117
VPSC	23.8	0.21	**21**
Wordle	14.6	0.31	220
Scanline Wordle	**13.6**	**0.34**	180

Table 3: The measures of the MDS dataset in detail.

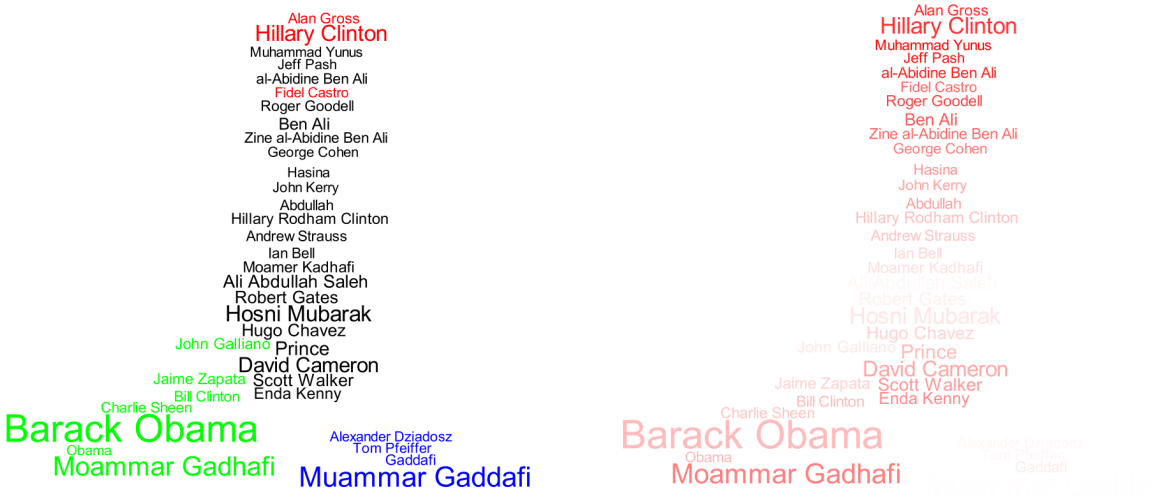


Figure 50: Left: Overlaps removed by Box2D. Right: Amount of displacement of the tags indicated red.

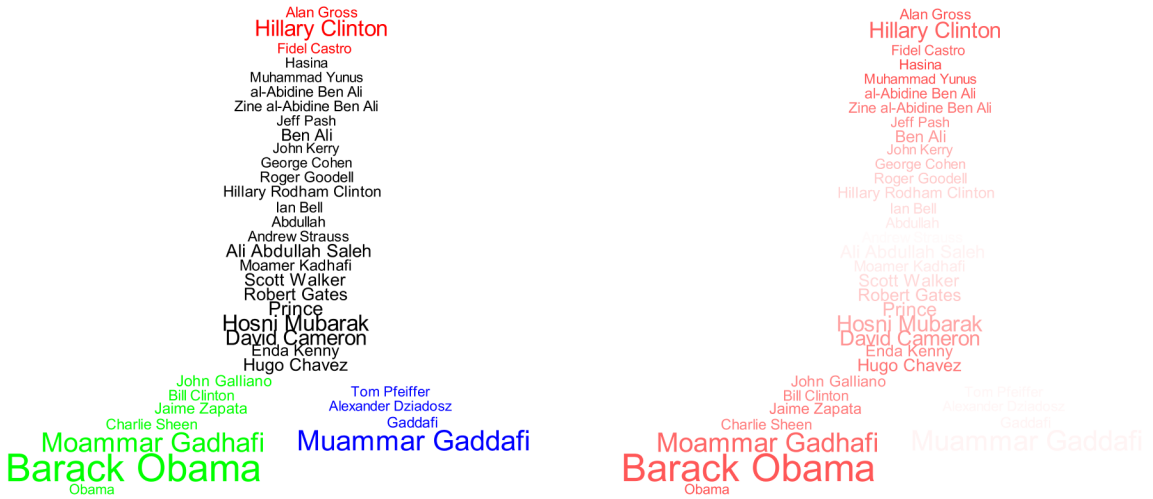


Figure 51: Left: Overlaps removed by VPSC. Right: Amount of displacement of the tags indicated red.

Figure 52: Left: Overlaps removed by Wordle. Right: Amount of displacement of the tags indicated red.

Figure 53: Left: Overlaps removed by VPSC. Right: Amount of displacement of the tags indicated red.

6 Conclusion

When trying to preserve the mental map with a layout adjustment algorithm, orthogonal ordering plays an important role. Algorithms like Box2D or VPSC primarily focus on preserving the orthogonal ordering between an initial and an adjusted layout.

However, when mainly focussing on orthogonal ordering, the resulting layout often gets spread out, especially when problems like the already mentioned stacking problem occur. The resulting layout looks different from the initial layout and as a result the mental map of the user is destroyed. Therefore, more metrics than just the orthogonal ordering have to be considered. When incorporating other metrics which describe the compactness and avoid the spreading out of the layout, better adjustments can be achieved.

The introduced *Scanline Wordle* layout adjustment algorithm has a good tradeoff between orthogonal ordering and compactness in terms of area occupancy and Euclidean distance which is a good balance in order to preserve the mental map. Also, it performs better than the other algorithms for artificial and real data which do not only have sparse overlaps, as has been shown.

7 Future Work

Weighted Elements

In the proposed overlap removal algorithm, all objects have been treated equally important, even when their sizes differed. When using weights for terms, the metrics and the layout algorithm should probably differ in some ways. First, the metrics would have to include the weights of the given terms. Placing an important term far aways from its initial position should result in an higher cost than doing the same with a less important term. In order to optimize the algorithm w.r.t. the metrics, the more important items would have to be priorized in the layouting strategy.

If no size of the objects is given, the question of how to map the weight to the size arises. The overlap removal algorithm could be tuned by using this information to generate a "better" layout.

Overlap Detection Performance

When generating the layout with Wordle or Scanline Wordle, many overlap tests have to be done. To improve the performance, bounding box tests prior to more costly shape intersections have already been used. Also, the first step is to always check wether or not the element which previously caused overlapping is still overlapping in order to prevent too many unnecessary tests. But still, the performance is highly dependent on the number of positions checked on the spiral. Also, checking two vector graphics for intersections is very costly.

However, if the application requires real-time calculations, like in a evolving tag cloud which depends on user input, speed is one of the most important quality measures. Keeping the already layouted elements as binary image might speed up the process. Also, more complex indexing structures, like a quadtree, could be used to minimize the number of overlap tests.

Sweepline PCA

When using the Scanline Wordle layout algorithm, a sweep line on either the x or y axis is used to order the elements by their x or y coordinates accordingly. The axis with the higher range of positions is chosen in order to keep the overall layout similar to the initial. When applying a Principal Component Analysis (PCA) on the positions, the first principal component has the highest variance possible. Using this PCA-axis to order the elements, there should be many cases where terms can instantly be positioned. With this, the Euclidean distance might be lower than currently.

List of Figures

1 Overlap removal in Box2D with forces . . . 3
2 Nongranular Box2D Bodies . . . 4
3 Granular Box2D body . . . 5
4 Constraint solver loop . . . 5
5 Constraint solver loop solution . . . 6
6 Wordle greedy layout alorithm . . . 6
7 Adaptive spiral sizes for Wordle . . . 7
8 Adaptive spiral point samples . . . 8
9 Rotation of terms in Wordle . . . 8
10 Example of Lloyd's method iteration . . . 9
11 Using cones to determine Voronoi areas . . . 10
12 Self movement of a limited Voronoi region . . . 11
13 Voronoi images in different iterations . . . 13
14 Optimal VPSC solution . . . 14
15 Area Occupancy as a metric of compactness . . . 16
16 Inversion count as number of inversions between two lists . . . 17
17 Separated Cluster: Initial . . . 18
18 Separated Cluster: Box2D . . . 19
19 Separated Cluster: VPSC . . . 19
20 Separated Cluster: Wordle . . . 19
21 Separated Cluster: Voronoi . . . 19
22 Overlapping Cluster: Initial . . . 21
23 Overlapping Cluster: Box2D . . . 21
24 Overlapping Cluster: VPSC . . . 21
25 Overlapping Cluster: Wordle . . . 22
26 Overlapping Cluster: Voronoi . . . 22
27 Uniform Distribution: Initial . . . 23
28 Uniform Distribution: Box2D . . . 24
29 Uniform Distribution: VPSC . . . 24
30 Uniform Distribution: Wordle . . . 24
31 Uniform Distribution: Voronoi . . . 24
32 Stacking problem in Box2D . . . 27
33 Greedy layout strategy in Scanline Wordle . . . 28
34 Central translation to preserve the mental map . . . 29
35 Map of Germany . . . 32
36 Map of Germany with overlaps removed by Box2D . . . 33
37 Map of Germany with overlaps removed by VPSC . . . 34
38 Map of Germany with overlaps removed by Wordle . . . 35
39 Map of Germany with overlaps removed by Scanline Wordle . . . 35
40 Map of England . . . 37
41 Map of England with overlaps removed by Box2D . . . 38

42 Map of England with overlaps removal by VPSC 39
43 Map of England with overlaps removed by Wordle 39
44 Map of England with overlaps removed by Scanline Wordle 40
45 Box2D stacking problem with scaling 41
46 Text section before semantic scaling. 42
47 Text section after semantic scaling and overlap removal 42
48 Decreased minimal scaling size with Scanline Wordle. 42
49 MDS Dataset: Initial 43
50 MDS Dataset: Box2D 44
51 MDS Dataset: VPSC 44
52 MDS Dataset: Wordle 45
53 MDS Dataset: VPSC 45

8 Bibliography

References

[1] Box2d. http://box2d.org, September 2011.

[2] Llyod's algorithm. http://en.wikipedia.org/wiki/Lloyds_algorithm, September 2011.

[3] Opencalais. http://www.opencalais.com/, September 2011.

[4] Wordle. http://www.wordle.net, September 2011.

[5] J Baumgarte. Stabilization of constraints and integrals of motion in dynamical systems. *Computer Methods in Applied Mechanics and Engineering*, 1(1):1–16, 1972.

[6] E Catto. *Iterative dynamics with temporal coherence*, pages 1–24. 2005.

[7] G Di Battista, P Eades, R Tamassia, and I G Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[8] Q. Du, V. Faber, and M. D. Gunzburger. Centroidal Voronoi tesselations: applications and algorithms. *SIAM Review*, 41(4):637–676, 1999.

[9] Qiang Du, Maria Emelianenko, and Lili Ju. Convergence of the lloyd algorithm for computing centroidal voronoi tessellations. *SIAM J. Numer. Anal.*, 44:102–119, January 2006.

[10] Tim Dwyer, Kim Marriott, and Peter J. Stuckey. Fast node overlap removal. In *In: Proc. 13th Int. Symp. on Graph Drawing (GD'05). Volume 3843 of LNCS. (2006) 153–164*, pages 153–164. Springer, 2005.

[11] Peter Eades and Roberto Tamassia. Algorithms for drawing graphs: An annotated bibliography. Technical report, Providence, RI, USA, 1988.

[12] Emden R. Gansner and Yifan Hu. Graph drawing. chapter Efficient Node Overlap Removal Using a Proximity Stress Model, pages 206–217. Springer-Verlag, Berlin, Heidelberg, 2009.

[13] John C. Handley, Anoop M. Namboodiri, and Richard Zanibbi. Document under-

standing system using stochastic context-free grammars. *Document Analysis and Recognition, International Conference on*, 0:511–515, 2005.

[14] Kunihiko Hayashi, Michiko Inoue, Toshimitsu Masuzawa, and Hideo Fujiwara. A layout adjustment problem for disjoint rectangles preserving orthogonal order. In *Proceedings of the 6th International Symposium on Graph Drawing*, GD '98, pages 183–197, London, UK, 1998. Springer-Verlag.

[15] Xiaodi Huang, Wei Lai, A. S. M. Sajeev, and Junbin Gao. A new algorithm for removing node overlapping in graph visualization. *Inf. Sci.*, 177:2821–2844, July 2007.

[16] Kyle Koh, Bongshin Lee, Bohyoung Kim, and Jinwook Seo. Maniwordle: providing flexible control over wordle. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1190–1197, 2010.

[17] Wanchun Li, Peter Eades, and Nikola Nikolov. Using spring algorithms to remove node overlapping, 2005.

[18] Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28:129–137, 1982.

[19] Kelly A. Lyons. Cluster busting in anchored graph drawing. In *Proceedings of the 1992 conference of the Centre for Advanced Studies on Collaborative research - Volume 1*, CASCON '92, pages 7–17. IBM Press, 1992.

[20] Kim Marriott, Peter Stuckey, Vincent Tam, and Weiqing He. Removing node overlapping in graph layout using constrained optimization. *Constraints*, 8:143–171, April 2003.

[21] Kazuo Misue, Peter Eades, Wei Lai, and Kozo Sugiyama. Layout Adjustment and the Mental Map. *Journal of Visual Languages & Computing*, 6(2):183–210, June 1995.

[22] M J Sabin and R M Gray. Global convergence and empirical consistency of the generalized lloyd algorithm. *IEEE Trans. Inf. Theor.*, 32:148–155, March 1986.

[23] Fernanda B. Viégas, Martin Wattenberg, and Jonathan Feinberg. Participatory visualization with wordle. *IEEE Trans. Vis. Comput. Graph.*, 15(6):1137–1144, 2009.