Content-Driven Layout for Visualization Design

Yngve S. Kristiansen
University of Bergen
Bergen, Norway
ykr088@uib.no

Laura Garrison
University of Bergen
Bergen, Norway
laura.garrison@uib.no

Stefan Bruckner
University of Bergen
Bergen, Norway
stefan.bruckner@uib.no

ABSTRACT

Multi-view visualizations are typically presented in a grid layout with elements positioned according to their bounding rectangles. These rectangles often contain unused white space. In cases where Tufte's Shrink Principle can be applied to reduce non-data-ink without impairing the communication of information, unused white space can be utilized for the placement of other elements. This is often done in manually "hand-crafted" layouts by designers. However, upon changes to individual elements, this design process has to be repeated. To reduce non-data-ink and repetitive manual design, we contribute a method for automatically turning a grid layout into a *content-driven* layout, where elements are positioned with respect to their contents. Existing approaches have explored the use of a force simulation in conjunction with proxy geometries to simplify collision handling for irregular shapes. Such customized force directed layouts are usually unstable, and often require additional constraints to run properly. In addition, proxy geometries become less accurate and effective with more irregular shapes. To solve these shortcomings, we contribute an approach for identifying central elements in an original grid layout in order to set up corresponding attractive forces. Furthermore, we utilize an image-based approach for collision detection and avoidance that works accurately for highly irregular shapes. We demonstrate the utility of our approach with three case studies.

CCS CONCEPTS

• **Human-centered computing → Visualization techniques**.

KEYWORDS

Multi-view visualizations, grid layout, force-directed layout.

ACM Reference Format:
Yngve S. Kristiansen, Laura Garrison, and Stefan Bruckner. 2022. Content-Driven Layout for Visualization Design. In *Proceedings of the 15th International Symposium on Visual Information Communication and Interaction (VINCI '22).* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Data-driven infographics and dashboards often have a small set of elements positioned by an underlying grid layout, which considers only bounding rectangles of individual elements. In this paper, we propose means for compacting the layout of designs that have (1) unused white space between contents of grid cells and (2) one or a few identifiable central elements. By Tufte's Shrink Principle [36], many data graphics can have their data-density increased and be reduced in area without loss of information or readability. One way to achieve this is to reduce the amount of non-data-ink, which occurs frequently in designs with underlying grid layouts. This process is frequently performed manually. However, manual designs become tedious to re-design upon changes to individual elements due to manual revisions or changes in the underlying data.

Consider a user placing a set of visual elements onto a canvas in order to create a layout. A concrete example of this would be a user designing a Tableau [34] dashboard by arranging elements into a grid layout, as shown in Figure 1a. In this layout, each element is contained by a bounding rectangle. Within these bounding rectangles, there may be much unused white space. However, a grid layout does not allow the user to utilize this space efficiently.

One alternative to a grid layout is a force-directed layout [14], which allows for flexible control of distances between elements. However, force-directed layouts are often designed to work with only simple shapes such as circles, and rarely support irregular shapes where other representations are necessary. Ali et al. [2] utilized a force layout wherein irregular shapes are represented by convex hulls. Such proxy geometries are approximations to complex, fine-grained details, and become less accurate with more irregular and complex shapes. For example, the marks of the scatter plot on the right in Figure 4 are difficult to describe geometrically without losing some fidelity. As another example, consider the lower concave region of the lungs in Figure 2. A convex hull would fail to properly capture this region. In this paper, we aim to enable the use of this white space by employing a force-directed layout that reflects the original grid layout topology by attracting peripheral elements towards central elements, avoiding content-to-content collisions with high precision, even for highly irregular shapes. We present a scheme for better preserving the original grid layout topology, which applies attractive forces only towards a small set of inferred central elements. We use a novel image-based approach for content-to-content repulsion that enables fine-grained control of distances between elements. With three case studies, we demonstrate how our approach successfully turns grid layouts with a high degree of unused white space into content-driven layouts. These layouts effectively utilize white space around irregular shapes, and are able to better capture the aesthetic qualities of a manual design.

2 RELATED WORK

The primary goal of a multi-view visualization layout is to position elements so that they convey information to the user as intended.

VINCI '22, August 16–28, 2022, Chur, Switzerland

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

Such layouts are typically designed manually, often with the help of a grid [12]. Grid-based layouts are frequently used in visualization software programs such as Tableau [34] and for other multi-view visualizations. Since then, many more sophisticated techniques for multi-view visualizations have been introduced. Our work is a continuation of this, as we provide a more content-driven alternative to the typically used grid layout.

Multi-view visualization layout techniques have been explored by several researchers. For example, Javed et al. [23] defined the space of composite visualization in terms of four operators: juxtaposition, superimposition, overloading, and nesting. Chen et al. [8] explored 360 visualizations and identified a set of composition and configuration patterns in multi-view visualizations. Also on the topic of dashboard design, Sarikaya et al. [30] contributed a design space across several dimensions, including functional design, purpose, audience, and data semantics. They further pointed out that dashboards are currently venturing into the realm of infographics. In this work, we aim to enable for such a transition from a traditional dashboard towards a more "infographics"-like dashboard by transforming its layout into a more compact "hand-crafted" version.

Automated layout techniques typically fall into one of two categories: machine learning techniques, and constraint based techniques [26]. These techniques often extend such quality measures to encompass higher-level concepts. Jahanian et al. [22] quantified concepts from art and aesthetics into a system for automatically designing magazine covers. This knowledge was further leveraged to create a recommendation system [21] for generating magazine covers by adhering to high-level intuitive cues such as "formal" or "sporty". In their layout process, they identify non-salient image segments of the main image, and use them as potential regions to place secondary content. Yang et al. [39] proposed a system to automatically generate layouts by leveraging expert-designed, topic-dependent templates and a computational framework for integrating and harmonizing high-level aesthetics and low-level image features. Moritz et al. [29] proposed Draco, a system for formalizing such design knowledge and guidelines, making them accessible to non-experts through a constraint-based Answer Set Programming language.

Techniques for **synthesis and optimization of grid layouts** are useful since they operate on an already widely used layout technique. Jacobs et al. [19] introduced an approach for adapting grid-based magazine layouts to different screen sizes. Xu et al. [38] proposed an interface for beautifying layouts by visualizing and editing relationships with sketching gestures. Sketchplorer [35] integrates sketch-based design with a real-time layout optimizer. It automatically infers the designer's task and searches for local and global improvements. Dayama et al. [10] presented a method for interactively transferring a layout of a single user interface design to another user interface. Li et al. [25] proposed the use of LayoutGAN, a generative adversarial network, to synthesize, model, and edit geometric relations between different 2D elements. Schrier et al. [32] presented a system for assembling documents from different sources into a single grid-based design, which automatically adapts to different viewing conditions and content selections.

Space usage optimization has been explored in the context of multi-view visualization as well as windowing management. A foundational idea behind such optimization is to reduce the loss of white space. Analogous to this idea, Albano and Sapuppo [1] explored heuristic methods for allocating irregular 2D shapes with a minimal amount of white space for reducing loss of physical fabric while cutting. Similarly, Bouganis and Shanahan [6] used computer vision and AI techniques to minimize white space in layouts with varying shapes on both regular and irregular surfaces. Ishak and Feiner [18] devised a technique dubbed content-aware layout to position several windows by taking their contents into account. Steinberger et al. [33] presented a dynamic window management technique which identifies coherent information that is then used as a basis for moving and scaling windows. Haraty et al. [16] proposed a genetic algorithm for optimizing multi-window layouts for specific tasks. Ishak and Feiner [18] devised a technique dubbed content-aware layout to position several windows by taking their contents into account. Zheng et al. [40] introduced an approach to generate high quality, content-aware magazine graphic designs by using a deep learning generative model trained on a large magazine-layout dataset. Effective screen space usage is becoming even more relevant with a myriad of different devices and screen sizes being used to dissect data. Kim et al. [24] characterized different responsive visualization strategies by analyzing 378 pairs of large screen and small screen visualizations. They identify implications for existing works as well as future work. Andrews and Smrdel [3] applied the principles of responsive web design, and leveraged these principles to make commonly used visualizations responsive. Motivated by interviewing journalists, and analyzing 231 responsive news visualizations, Hoffswell et al. [17] proposed a prototype system for previewing and editing multiple visualization versions simultaneously. While such works focus on optimization of space usage, they typically consider elements only by their bounding rectangles, rather than contents. Our work shares the goal of efficient use of space, but achieves it for cases where it is more desirable to have a compact layout that adheres precisely to the contents of its elements.

The common **force-directed layout approach** [14] is often used alone or in conjunction with other constraints to achieve highly flexible graph layouts. Force-directed layouts have a wider range of applications, including visualizing biological pathways [15], improving Euler diagrams [28], rendering Lombardi-style graphs [9], targeting network spatialization [20], and rapidly visualizing large networks [7]. Dengler et al. [11] used a generalization of a simple force-layout to generate diagram layouts satisfying geometric and aesthetic/perceptual constraints. We are inspired by the work of Ali et al. [2] who proposed a tool for creating blueprints to integrate interactive illustrations into one layout. However, their approach represented elements by their convex hulls, limiting its ability to tightly arrange irregular shapes (for example, the concave region of the lungs in Figure 2). Our method uses an image-based, rather than geometric, approach to deriving Euclidean content-to-content distances between elements, which does not suffer from these drawbacks.

3 CONTENT-DRIVEN LAYOUT

Layouts used in the context of multi-view visualizations typically position elements by their bounding rectangles, rather than contents. In a *content-driven* layout, elements are positioned by their

contents, i.e., pixels that are not white space. A content-driven layout can make better use of previously unused white space, resulting in a potentially more visually pleasant, compact and "hand-crafted" appearance as seen in Figure 3. The main goal of our method is to enable the automatic transformation of a grid layout to a content-driven layout. This is achieved by generating a force-directed layout, with forces derived from the original grid layout.

3.1 Terminology

A **grid layout** (Figure 1a) is a layout where a rectangular space is recursively subdivided horizontally or vertically. A single **element** contains visual content and white space, while **content** excludes white space. Each grid layout corresponds to a hierarchy as seen in Figure 1b. If several elements in the grid layout share the same strip of space, they have the same parent node in the corresponding hierarchy, where the left-to-right order of nodes correspond to the order in which elements appear in the grid layout. In this hierarchy, non-leaf nodes have a **flow** that controls the order and direction in which their children are placed, which is either vertical left-to-right, or horizontal top-to-bottom. Consider how elements C, D, E in Figure 1a correspond to nodes in the hierarchy in Figure 1b. These three nodes also share the same parent (node 3 in Figure 1b), which flows vertically top-to-bottom, while A is a child of the root node (node 1 in Figure 1b), which also flows vertically top-to-bottom. Adjacency relationships between elements in the grid layout are referred to as the **topology** of the layout. For example, in Figure 1a we see that elements C, D, and E are to the left, and appear in order from top to bottom, while B, I, and J are in the middle, and element A is on the top. This topology can be concisely expressed by an extracted **adjacency graph** (Figure 1c), where each element in the grid layout is represented as a node positioned at its original center of gravity, and each adjacency between two elements in the grid layout is represented as a link between two nodes.

3.2 Overview

On a high level, our method follows a pipeline composed of the following steps. First, we derive adjacency relationships from the underlying tree structure of the grid layout (Figure 1b) and store them as an adjacency graph (Figure 1c). This adjacency graph is then used to identify a set of nodes (Figure 1d), which correspond to the visually most central elements in the original layout (Figure 1a, elements B and I). Attractive forces are set up such that in a force simulation, elements are pulled only *towards* these central elements. Image-based repulsive forces between each distinct pair of elements are used for content-to-content repulsion. With these attractive and repulsive forces, a force simulation is started, where each element is initially placed according to its original grid layout position. The execution of the force simulation transforms the grid layout into a content-driven layout.

3.3 Attractive Forces

Content-to-content attraction ensures that elements gravitate towards each other. Visually, it is easy to see that a grid layout has a small set of central elements. To represent the topology of the original grid layout, we model attractive forces so that they are all directed *towards* these central elements. To achieve this, we derive a graph from the original grid layout where each link represents the adjacency relationship between two elements in the grid layout. From this adjacency graph, we infer a minimal set of central nodes. Finally, we apply attractive forces that pull elements towards central elements.

Algorithm 1 Inferring adjacency graph

1: **procedure** POS($path$)
2: $ancestorFlow$ = flow of first node in path
3: $path$' = $path$ without its two first nodes
4: nlp = nodes in $path$' with $parent.flow = ancestorFlow$
5: return '$only$' if $nlp.length = 0$
6: return '$first$' if $\forall n \in nlp : index(n) = 0$
7: return '$last$' if $\forall n \in nlp : index(n) = |\text{siblings}(n)|$
8: return '$middle$'
9: **procedure** ISADJACENT(a, b)
10: if a and b are siblings: return $true$
11: $l \leftarrow \text{LCA}(a, b)$ ▷ Lowest common ancestor
12: a', b' = ancestors of a and b that are children of l
13: if index(a') > index(b'): return isAdjacent(b, a)
14: if index(b') - index(a') > 1: return $false$
15: $path_a, path_b$ = path from l to a, and l to b
16: return $\text{pos}(path_a) \in \{'last', 'only'\} \wedge \text{pos}(path_b) \in \{'first', 'only'\}$

We define the adjacency graph by a conditional that determines adjacency between two elements a and b. Throughout this paragraph, we refer to lines in Algorithm 1, and consider the nodes B and F in Figure 1 as examples of a and b, respectively. Adjacency between non-sibling nodes is determined by first finding their LCA (lowest common ancestor) in the grid hierarchy (line 10). Next, we extract the paths $path_a$ and $path_b$ between the nodes and their LCA (line 11), which would correspond to the paths consisting of node 4 between B and 2, and 5 between F and 2. Now, the goal is to find out if there are no other leaf nodes *between* the two possibly adjacent nodes. Thus, we check how $path_a$ and $path_b$ are positioned in relation to each other by using the subroutine *pos* (line 1). This subroutine checks the order of appearance of nodes with respect to the flow of the LCA. For example, since the flow of the LCA (node 2) is horizontal, and there are no other horizontally flowing nodes *between* B and 2, or F and 2, *pos* will in both cases return '$only$'. If there was a horizontally flowing node on the path from F to 2, F would have to be the horizontally first or only leaf node to be in contact with B. Conversely, B would have to be the horizontally last or only leaf node in the case of a horizontally flowing node between B and 2. Thus, two nodes are adjacent if *pos* evaluates to '$last$' or '$only$' for $path_a$, and '$first$' or '$only$' for $path_b$ as seen on line 16.

From the adjacency graph, we can now determine the most central elements. For every node n in the adjacency graph, we count how many of its immediate neighbors are unvisited, and store this number as $n.a$ as seen on line 6 of Algorithm 2. Furthermore, we count how many of its neighboring nodes are connected to an unvisited node, and store this number as $n.b$ (Algorithm 2, line 7). We then identify the highest a-value among all non-central nodes (Algorithm 2, line 9), and find all nodes with this a-value (Algorithm 2, line 10). From these nodes, we find the highest b-value,

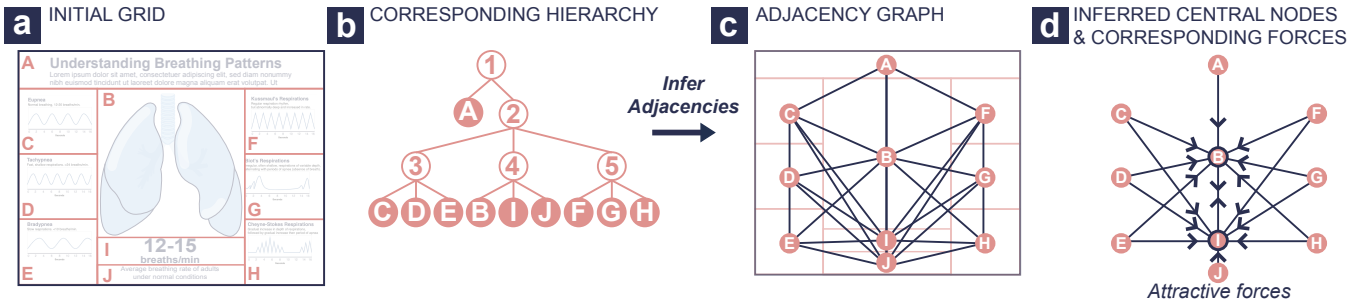

Figure 1: Here we see the steps for generating a set of attractive forces from an initial grid layout (a), which corresponds to a grid layout hierarchy (b). From this hierarchy (b) we infer all adjacent elements in the grid layout (a) using Algorithm 1, and store them as a neighborhood graph (c). This neighborhood graph is again used to determine which elements are central in the original layout (a). In this case, elements B and I are inferred to be central nodes (d). This results in attractive forces directed only *towards* B and I, indicated by the direction of the arrows along the links (d).

Algorithm 2 Identifying central elements

```
1:  procedure FINDCENTRALNODES(N)
2:  C ← []                                          ▷ central nodes
3:  V ← []                                          ▷ visited nodes
4:  until all nodes in N are in C or V, do:
5:    for each node n : n ∈ N, do:
6:      n.a ← number of unvisited neighbors of n
7:      n.b ← number of neighbors connected to unvisited node
8:    N' ← V if |V| ≠ ∅, otherwise N
9:    a_max ← max({ n.a | n ∈ N' })                 ▷ Greatest a
10:   A ← { n | n ∈ V ∧ n.a = a_max }               ▷ Nodes with greatest a
11:   b_max ← max({ n.b | n ∈ A })                  ▷ Greatest b
12:   for each node n ∈ A where n.b = b_max, do:
13:     add n to C                                  ▷ n is central
14:     add all neighbors of n to V                 ▷ neighbors of n are visited
```

as shown on line 11 of Algorithm 2. Finally, we select only the nodes with the highest identified a-value, and associated highest b-value (Algorithm 2, line 12), which are tagged as central (Algorithm 2, line 13). Furthermore, the neighbors of each central node are tagged as visited on line 14 of Algorithm 2.

Based on the computed central nodes, we can now apply attractive forces drawing corresponding peripheral elements towards central elements, and each central element towards all other central elements. In other words, no elements are attracted *towards* peripheral elements, and bidirectional attractive forces are set up between all central elements. The **directionality** of a force attracting element B towards element A is defined as the normalized vector going from the center of gravity of B, to the center of gravity of A. We use standard spring forces based on Hooke's law with a constant **strength**.

3.4 Repulsive Forces

Content-to-content repulsion prevents elements from overlapping. The first step towards achieving this is to consider each element by its content rather than its bounding rectangle. In existing approaches, irregular shapes are often simplified to geometric shapes, which are then used to compute content-to-content distances and overlaps. For example, Ali et al. [2] used convex hulls to compute content-to-content distances and collisions. Such geometry-based approaches become less reliable and more difficult to handle with irregular shapes. For example, consider the image of a lung in Figure 2. Using a convex hull around this fails to represent the gap between the two lungs. Hence, we use an image-based approach to detect and avoid overlapping contents. We achieve accurate content-to-content distances by utilizing the Euclidean distance transform [4], which we compute for every element by using Meijster's algorithm [27]. The distance transform of an element A is denoted as dt_A, and encodes in every pixel the Euclidean distance to the nearest content pixel. We compute the distance transform from a binary image of the original content, where 1 is content and 0 is white space.

Repulsive forces are applied between all elements. These forces are active only if elements have overlapping bounding rectangles. If two elements A and B have intersecting bounding rectangles at the region $A \cap B$, they may also have overlapping contents. To compute the distance between the contents of the two views, we first consider the distance transforms of A and B, dt_A and dt_B. Then, we clip out the region $A \cap B$ from the distance transforms of A and B, giving us the images dt'_A and dt'_B. These clipped distance transforms are then summed together pixel by pixel, giving the content-to-content distance at the smallest pixel as follows:

$$d_{AB} = \min(dt'_A + dt'_B) \tag{1}$$

The repulsive force pushes the content of element B away from the content of element A. The **directionality** of repulsion is determined by the gradient of the distance transform. To compute the directionality, we consider the distance transform gradient of A, at the region of intersection $A \cap B$, denoted $\nabla dt'A$. The directionality of repulsion is the normalized sum of vectors from $\nabla dt'_A$.

In practice, it is often desirable to have a certain margin M around the content of an element. This requires correspondingly enlarging the bounding rectangle of an element to avoid issues when the content is close to its borders. The **repulsive strength** between two elements A and B is then computed as:

$$\rho_{AB} = \frac{1}{\max(d_{AB} - M, \theta)} \tag{2}$$

where θ acts as a lower threshold for the denominator of the expression, avoiding near infinite strengths as $d_{AB} - M$ approaches zero.

4 IMPLEMENTATION

We implemented our approach by using Vue.js and TypeScript for rendering the layout elements, and D3.js [5] for running the force simulation. The attractive and repulsive forces are implemented as custom forces in D3's force-directed layout implementation. Layout elements are reactively linked to underlying layout objects, which are updated in each step of the layout simulation.

The underlying grid layout is specified as a recursive JSON object corresponding to the underlying tree structure outlined in Section 3. Non-leaf nodes have the following properties: (1) **flow**, which is either vertical or horizontal, and **children**, a list of children. Leaf nodes have a **source** property which may be an image link, or a Vega-Lite [31] specification string which enables the easy integration of a rich set of different types of visualizations.

The most expensive part of our algorithm is finding the content-to-content repulsion between nodes. More specifically, adding together the distance transforms at the region of intersection, and finding the smallest pixel is most time-consuming. We currently use a straight-forward CPU implementation using Javascript's Float64Array to represent the underlying distance transforms, so a GPU-based approach could lead to significant time reductions. The performance of this operation is also highly dependent on the resolution of the distance transform. To illustrate this impact, we present the performance of our algorithm with two different distance transform resolutions: (1) $\max(width, height) = 50$, and (2) $\max(width, height) = 200$. We generated the lungs example in Figure 2 using the Mozilla Firefox browser, on a machine with 32GB RAM and a Intel Core i7-7700K CPU @ 4.20GHz. With both resolutions, convergence was reached after 35 iterations of the force layout, which took 905 and 1207 milliseconds for resolutions 50 and 200, respectively. On average, each iteration took 25.8 milliseconds with the resolution of 200, and 18 milliseconds with a resolution of 50 (in both cases, this includes rendering time). In our experiments, we found that even when using resolutions significantly lower than rendering resolutions, impacts on the final results were negligible with large-scale structure shapes such as the lungs in Figure 2. However, a higher resolution was required to capture finer aspects such as very small bubbles in the bubble plot in Figure 4.

5 CASE STUDIES

In this section we illustrate the value of our approach in three case studies in collaboration with a designer who is a coauthor of this paper. Each case study begins with the project brief that a designer or data journalist would typically receive at the beginning of a project. We then discuss the ideation phase, where the designer develops layout concepts, and the subsequent challenges faced in typical production tools such as Tableau or Adobe Illustrator. We then compare the functional capabilities of these typical production tools against our content-driven layout algorithm. For all case studies we use styling from the Vega cook book repository[1].

[1]https://github.com/aezarebski/vegacookbook

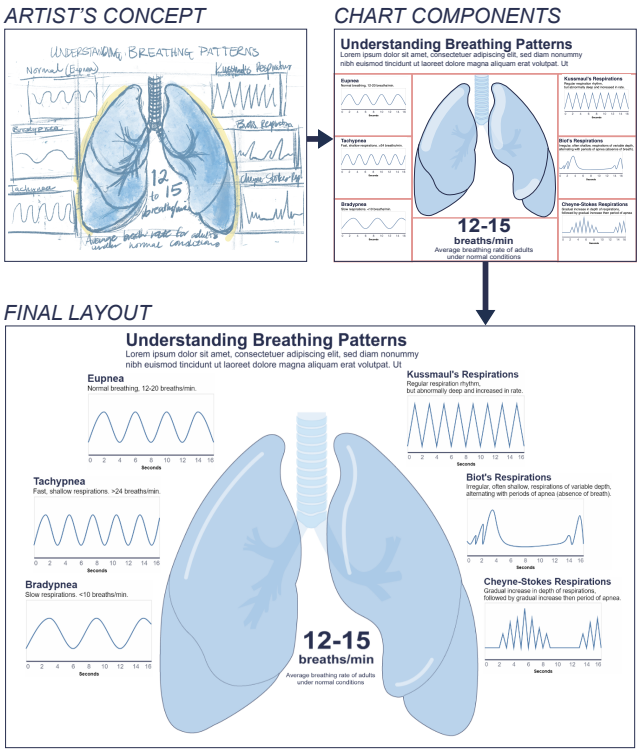

Figure 2: Using our content-driven layout approach, the artist is able to achieve a composition where the six respiratory patterns are arrayed around the margins of the lungs.

A demonstration video illustrating our approach and these case studies is available at: https://tinyurl.com/y652kt6n.

5.1 Respiration Patterns

We begin with a case study illustrating a relatively simple white space optimization challenge that is not possible to achieve in standard visualization tools, such as in Tableau. The design brief for this scenario is to design an infographic describing six common breathing patterns as seen in the output of a ventilator. This is intended for medical staff to use as a study aid, with an engaging vector graphic of the lungs and additional text describing the normal rate of respiration. The lungs are intended as the center piece, with the six respiration patterns arrayed around this central graphic.

Per the design brief, the designer places the lung anatomy centrally and as the largest graphical element in the figure. To add visual interest to the infographic, they array the six respiration patterns in a slightly out-of-grid layout to fan around the outer borders of the lungs, as per the artist's concept sketch in Figure 2. Additionally, the text stating the average normal breathing rate (12–15 breaths per minute) per the brief also requires visual emphasis, and the gap where the heart would normally rest in the white space between the lungs is an ideal position for its optimization of white space and visual interest. However, such a placement is impossible to implement in Tableau, which uses a grid-based design ignoring white space around the graphical assets, per Figure 2, top

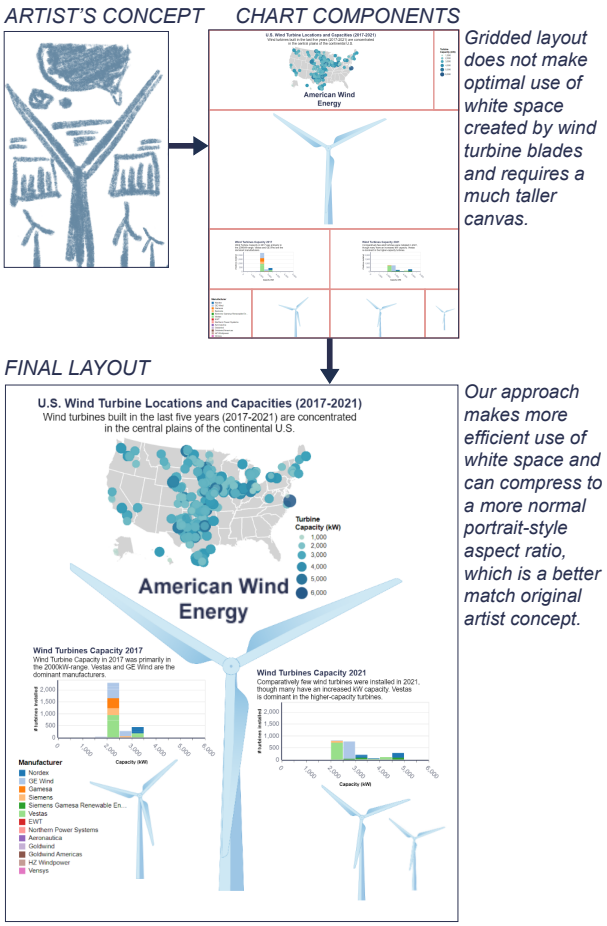

Figure 3: Our content-driven layout enables plots and text elements to array around the large, central wind turbine to optimize use of white space and more closely match the artist's intended layout.

right. These adjustments would typically need to be implemented in Adobe Illustrator or similar software. While creating such a layout is reasonably low-effort once, placement and alignment becomes tedious with multiple design iterations, particularly if design elements change. For example, an interactive version of this infographic may require the average respiration rate text to change with selection of particular respiration patterns. This requires minor layout tweaks in each iteration that rapidly become tedious for the designer. Our method enables the graphs of respiration patterns to align smoothly along the margins of the lungs in the central graphic. It furthermore is able to move the average respiration text into the white space between the lungs to add visual interest and emphasis to this information.

5.2 Wind Turbine Distribution in the US

This case study illustrates the value of our method when creating layouts that use the white space surrounding highly irregular-shaped assets. In this instance, the designer has been tasked with creating a visually-engaging poster infographic explaining the distribution of wind turbines across the US. The brief requires hand-drawn vector graphics of wind turbines alongside charts describing wind turbine geographic distribution and power capacity, as well as charts describing changes in power capacity and dominant manufacturers from 2017 to 2021.

Wind turbines are large, irregularly-shaped objects, with large amounts of white space between the turbine blades of the central, large wind turbine that anchors the other design elements of the infographic. To add visual interest and economize white space, the designer wishes to nest the data visualizations in these white spaces, as in the artist's concept shown in Figure 3. Such use of white space normally requires manual layout creation in Adobe Illustrator or similar artistic programs. The closest achievable layout in a grid-based system is illustrated in the top right image of Figure 3. There are several instances of inefficient use of white space in this layout. First, the title of the infographic cannot efficiently nest between the blades of the central turbine. Second, the width of the bounding box around the central turbine is dictated by the span of the turbine blades, where there is insufficient space on either side to draw the bar charts and companion smaller wind turbine graphical elements. These four elements are not able to nest below the turbine blades and thereby make more efficient use of the white space in the lower half of the infographic. The canvas must be taller to accommodate the two smaller windmills and bar charts, which are pushed below the central wind turbine element.

Our content-driven approach generates more compact layouts for positioning irregularly-shaped graphics with a graphically-interesting use of white space underneath the large wind turbine blades. The infographic title, "American Wind Energy" tucks into the space immediately above the central wind turbine, which enables enlargement of the wind turbine distribution map. The two bar charts describing wind turbine capacity and manufacturer for the years 2017 and 2018, with their companion graphical wind turbine elements, can move close to either side of the turbine pole, which enables the entire canvas to take on the standard portrait aspect ratio dimensions that the designer originally intended.

5.3 Health vs. Wealth in the Countries of the World

Our final case study considers the optimization of white space under two changing conditions in a visualization: (1) changing the accompanying image element in a visualization and (2) changing the scales of the axes of a plot that leads to a different distribution of white space. This design brief is for a visualization plotting the health against wealth of 187 countries of the world, from the GapMinder dataset[2]. To add visual interest, the brief requests each data point to include a map of the country.

This design is centered around a bubble plot that describes the correlation of countries' health and wealth. Each bubble represents a country, and its size is encoded to the population of that country. Hovering over a bubble reveals a tooltip of information for that country, as well as a map of the country. The designer furthermore wishes to have this map element appear as close to the associated country bubble as possible, without obscuring surrounding data

[2]https://www.gapminder.org/data/

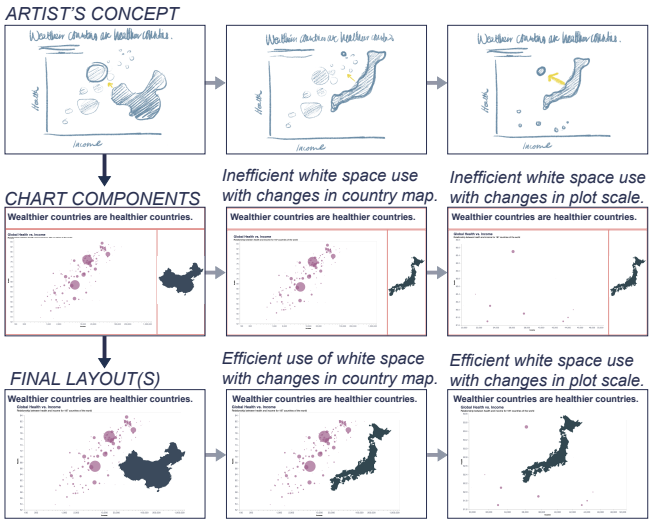

Figure 4: Concept for visualization of health vs. wealth of countries of the world. While a grid layout is unable to place a country map element near the data points, our content-driven approach places the map element in an optimal position. This concept shows efficient use of white space in (1) changing elements and (2) changing white space.

points in the bubble plot. A complication to this brief is that the plot scales can be adjusted to zoom in to different regions of the plot, which requires a different layout for the map element depending on the changing white space.

The designer's concept is shown in the top of Figure 4 for inspection of China and Japan, and a subsequent alteration of the plot scale to look closer at only the healthiest and wealthiest countries, including Japan. The design in these three instances, first with the change of image element and second with the change of plot scale and subsequent bubble positioning, takes advantage of the extensive white space around the bubble plot to nest maps closely around the data elements. Unfortunately, this layout is not achievable in a standard grid-based layout system, which is illustrated in the middle row of Figure 4. Here, the maps must be much smaller and further away from their associated data point. This is an inefficient use of white space, and forces the viewer to look back and forth between associated elements.

Our content-driven approach enables the country map element to slide in near the data point of interest to reduce white space in the visualization, and to reduce the number of places the user must focus their attention on in the visualization. In the first scenario at the bottom left two images of Figure 4, the differently-shaped country maps of China and Japan each are able to position efficiently in the white space in the lower right of the bubble plot. When adjusting the plot scales to show only the healthiest and wealthiest countries (bottom rightmost), our algorithm positions the Japan map element more optimally in the upper right of the bubble plot. This case study thus demonstrates our algorithm's capabilities both in conditions of (1) changing elements and (2) changing white space.

6 DISCUSSION AND LIMITATIONS

As demonstrated in the presented case studies, our approach is capable of transform a grid layout that makes inefficient use of white space into a content-driven layout that can closely mirror the artist's original design concept and more efficiently use the white space in the visualization. Our approach is targeted at scenarios with relatively few elements (i.e., tens), which is typical for common infographics, and hence is not well-suited for the layout of large data collections [13].

While the method presented in this paper primarily focuses on turning grid layouts into content-driven layouts, it could greatly benefit from being combined with existing works and approaches. Our force-directed layout setup could be used on other initial layouts with irregular shapes, as long as it is possible to infer the layout topology as a graph. Our prototype starts from a fully-specified grid layout and specific screen size, which introduces some instability to the layout. However, using interaction to incrementally place or move shapes could help tweak and overcome such instabilities.

Although we found that our approach delivers good results even with distance transforms computed at lower resolutions, there may be instances where this fails to take into account details that may be relevant for the final layout. In such cases, an additional preprocessing step could be used (e.g., a low-pass filter) to make sure that all important features are captured. Alternatively, or additionally, the input to the distance transform could be adapted according to the semantics of individual components of the visualization, e.g., by giving higher priority to data-encoding pixels compared to legends or other decorations. Such extensions could be easily integrated into our approach by automatically applying different styles for layout and display purposes to the Vega-Lite specification. Similarly, object detection techniques could be leveraged to identify salient regions of images or charts and separate it from the background, before generating the distance transform. Furthermore, image-space edits could be done to the image before the distance transform is computed. For example, drawing a line on the right side of an image before computing the distance field would create a hard boundary on the right side of that image in the layout simulation, and such manipulations could be used as additional alignment guides. Other types of interactions could also be integrated in order to further increase the degree of design freedom. While currently our approach is based on an automatic identification of central elements, this initial selection could be modified by the user in order to provide a more tailored user experience as discussed by Tyagi et al. [37].

While our current prototype implementation already allows for basic interactions when supplied with corresponding Vega-Lite specifications, we do not yet support fully dynamic content (e.g., interactive filtering or cross-linking between multiple views), partly due to the fact that it has proven difficult to implement event translation consistently across different browsers. However, we plan to extend this functionality in the future. We plan to evaluate our approach through a user study or a larger set of existing grid layouts and element types. Here, in particular, it will be interesting to study differences related to the design expertise of individual users.

7 CONCLUSION

In this paper we presented an approach for turning an existing grid layout into a content-driven layout, wherein elements are positioned by their contents rather than their proxy bounding geometries. Our method parses and transforms the original grid layout topology into a smaller graph which informs the selective application of attractive forces and leverages distance transforms to repulse elements based on their content to enable better space utilization. Furthermore, we presented three case studies demonstrating the effectiveness and versatility of our algorithm. In the future, our approach to repulsing irregular shapes can be applied to other use cases, in combination with other image-based techniques.

REFERENCES

[1] Antonio Albano and Giuseppe Sapuppo. 1980. Optimal allocation of two-dimensional irregular shapes using heuristic search methods. *IEEE Transactions on Systems, Man, and Cyfbernetics* 10, 5 (1980), 242–248. https://doi.org/10.1109/TSMC.1980.4308483

[2] Kamran Ali, Knut Hartmann, Georg Fuchs, and Heidrun Schumann. 2008. Adaptive layout for interactive documents. In *Proc. International Symposium on Smart Graphics*. 247–254. https://doi.org/10.1007/978-3-540-85412-8_24

[3] Keith Andrews and Ales Smrdel. 2017. Responsive data visualisation. In *Proc. EuroVis (Posters)*. 113–115. https://doi.org/10.2312/eurp.20171182

[4] Gunilla Borgefors. 1986. Distance transformations in digital images. *Computer vision, graphics, and image processing* 34, 3 (1986), 344–371. https://doi.org/10.1016/S0734-189X(86)80047-0

[5] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D^3: data-driven documents. *IEEE Transactions on Computer Graphics and Visualization* 17, 12 (2011), 2301–2309. https://doi.org/10.1109/TVCG.2011.185

[6] Alexandros Bouganis and Murray Shanahan. 2007. A vision-based intelligent system for packing 2-D irregular shapes. *IEEE Transactions on Automation Science and Engineering* 4, 3 (2007), 382–394. https://doi.org/10.1109/TASE.2006.887158

[7] Govert G. Brinkmann, Kristian F.D. Rietveld, and Frank W. Takes. 2017. Exploiting GPUs for fast force-directed visualization of large-scale networks. In *Proc. International Conference on Parallel Processing*. 382–391. https://doi.org/10.1109/ICPP.2017.47

[8] Xi Chen, Wei Zeng, Yanna Lin, Hayder Mahdi Al-Maneea, Jonathan C. Roberts, and Remco Chang. 2020. Composition and Configuration Patterns in Multiple-View Visualizations. *CoRR* abs/2007.15407 (2020). https://doi.org/10.48550/arXiv.2007.15407

[9] Roman Chernobelskiy, Kathryn I Cunningham, Michael T Goodrich, Stephen G Kobourov, and Lowell Trott. 2011. Force-directed Lombardi-style graph drawing. In *Proc. International Symposium on Graph Drawing*. 320–331. https://doi.org/10.1007/978-3-642-25878-7_31

[10] Niraj Ramesh Dayama, Simo Santala, Lukas Brückner, Kashyap Todi, Jingzhou Du, and Antti Oulasvirta. 2021. Interactive layout transfer. In *Proc. International Conference on Intelligent User Interfaces*. 70–80. https://doi.org/10.1145/3397481.3450652

[11] Edmund Dengler, Mark Friedell, and Joe Marks. 1993. Constraint-driven diagram layout. In *Proc. IEEE Symposium on Visual Languages*. 330–335. https://doi.org/10.1109/VL.1993.269619

[12] Steven Feiner. 1988. A grid-based approach to automating display layout. In *Proc. Graphics Interface*. 192–197.

[13] Steffen Frey. 2022. Optimizing Grid Layouts for Level-of-Detail Exploration of Large Data Collections (preprint). (2022). https://freysn.github.io/papers/ldg.pdf

[14] Thomas MJ Fruchterman and Edward M Reingold. 1991. Graph drawing by force-directed placement. *Software: Practice and Experience* 21, 11 (1991), 1129–1164. https://doi.org/10.1002/spe.4380211102

[15] Burkay Genc and Ugur Dogrusoz. 2003. A constrained, force-directed layout algorithm for biological pathways. In *Proc. International Symposium on Graph Drawing*. 314–319. https://doi.org/10.1007/978-3-540-24595-7_29

[16] Mona Haraty, Syavash Nobarany, Steve DiPaola, and Brian D. Fisher. 2009. AdWiL: adaptive windows layout manager. In *Proc. International Conference on Human Factors in Computing Systems*. 4177–4182. https://doi.org/10.1145/1520340.1520636

[17] Jane Hoffswell, Wilmot Li, and Zhicheng Liu. 2020. Techniques for flexible responsive visualization design. In *Proc. CHI Conference on Human Factors in Computing Systems*. 1–13. https://doi.org/10.1145/3313831.3376777

[18] Edward Waguih Ishak and Steven Feiner. 2007. Content-Aware Layout. In *Proc. CHI Extended Abstracts on Human Factors in Computing Systems*. 2459—-2464. https://doi.org/10.1145/1240866.1241024

[19] Charles Jacobs, Wilmot Li, Evan Schrier, David Bargeron, and David Salesin. 2003. Adaptive grid-based document layout. *ACM Transactions on Graphics* 22, 3 (2003), 838–847. https://doi.org/10.1145/882262.882353

[20] Mathieu Jacomy, Tommaso Venturini, Sebastien Heymann, and Mathieu Bastian. 2014. ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software. *PloS one* 9, 6 (2014), e98679. https://doi.org/10.1371/journal.pone.0098679

[21] Ali Jahanian, Jerry Liu, Qian Lin, Daniel Tretter, Eamonn O'Brien-Strain, Seungyon Claire Lee, Nic Lyons, and Jan Allebach. 2013. Recommendation system for automatic design of magazine covers. In *Proc. International Conference on Intelligent User Interfaces*. 95–106. https://doi.org/10.1145/2449396.2449411

[22] Ali Jahanian, Jerry Liu, Daniel R Tretter, Qian Lin, Niranjan Damera-Venkata, Eamonn O'Brien-Strain, Seungyon Lee, Jian Fan, and Jan P Allebach. 2012. Automatic design of magazine covers. In *Proc. Imaging and Printing in a Web 2.0 World III*. 114–121. https://doi.org/10.1117/12.914596

[23] Waqas Javed and Niklas Elmqvist. 2012. Exploring the Design Space of Composite Visualization. In *Proc. IEEE PacificVis*. 1–8. https://doi.org/10.1109/PacificVis.2012.6183556

[24] Hyeok Kim, Dominik Moritz, and Jessica Hullman. 2021. Design patterns and trade-offs in responsive visualization for communication. *Computer Graphics Forum* 40, 3 (2021), 459–470. https://doi.org/10.1111/cgf.14321

[25] Jianan Li, Jimei Yang, Aaron Hertzmann, Jianming Zhang, and Tingfa Xu. 2019. Layoutgan: Generating graphic layouts with wireframe discriminators. *arXiv preprint arXiv:1901.06767* abs/1901.06767 (2019). https://doi.org/10.48550/arXiv.1901.06767

[26] Simon Lok and Steven Feiner. 2001. A survey of automated layout techniques for information presentations. *Proc. Smart Graphics* 2001 (2001), 61–68.

[27] Arnold Meijster, Jos BTM Roerdink, and Wim H Hesselink. 2002. A general algorithm for computing distance transforms in linear time. In *Proc. Mathematical Morphology and its Applications to Image and Signal Processing*. 331–340. https://doi.org/10.1007/0-306-47025-X_36

[28] Luana Micallef and Peter Rodgers. 2014. eulerforce: Force-directed layout for Euler diagrams. *Journal of Visual Languages & Computing* 25, 6 (2014), 924–934. https://doi.org/10.1016/j.jvlc.2014.09.002

[29] Dominik Moritz, Chenglong Wang, Greg L Nelson, Halden Lin, Adam M Smith, Bill Howe, and Jeffrey Heer. 2018. Formalizing visualization design knowledge as constraints: Actionable and extensible models in draco. *IEEE Transactions on Computer Graphics and Visualization* 25, 1 (2018), 438–448. https://doi.org/10.1109/TVCG.2018.2865240

[30] Alper Sarikaya, Michael Correll, Lyn Bartram, Melanie Tory, and Danyel Fisher. 2018. What do we talk about when we talk about dashboards? *IEEE Trans. Visualization and Computer Graphics* 25, 1 (2018), 682–692. https://doi.org/10.1109/TVCG.2018.2864903

[31] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2016. Vega-lite: A grammar of interactive graphics. *IEEE Transactions on Computer Graphics and Visualization* 23, 1 (2016), 341–350. https://doi.org/10.1109/TVCG.2016.2599030

[32] Evan Schrier, Mira Dontcheva, Charles Jacobs, Geraldine Wade, and David Salesin. 2008. Adaptive layout for dynamically aggregated documents. In *Proc. International Conference on Intelligent User Interfaces*. 99–108. https://doi.org/10.1145/1378773.1378787

[33] Markus Steinberger, Manuela Waldner, and Dieter Schmalstieg. 2012. Interactive self-organizing windows. 31, 2pt3 (2012), 621–630. https://doi.org/10.1111/j.1467-8659.2012.03041.x

[34] Chris Stolte, Diane Tang, and Pat Hanrahan. 2008. Polaris: a system for query, analysis, and visualization of multidimensional databases. *Commun. ACM* 51, 11 (2008), 75–84. https://doi.org/10.1109/2945.981851

[35] Kashyap Todi, Daryl Weir, and Antti Oulasvirta. 2016. Sketchplore: Sketch and explore with a layout optimiser. In *Proc. ACM Conference on Designing Interactive Systems*. 543–555. https://doi.org/10.1145/2901790.2901817

[36] Edward R. Tufte. 1986. *The Visual Display of Quantitative Information*. Graphics Press, USA. https://doi.org/10.5555/33404

[37] Anjul Tyagi, Jian Zhao, Pushkar Patel, Swasti Khurana, and Klaus Mueller. 2022. Infographics Wizard: Flexible Infographics Authoring and Design Exploration. https://doi.org/10.48550/ARXIV.2204.09904

[38] Pengfei Xu, Hongbo Fu, Takeo Igarashi, and Chiew-Lan Tai. 2014. Global beautification of layouts with interactive ambiguity resolution. In *Proc. ACM Symposium on User Interface Software and Technology*. 243–252. https://doi.org/10.1145/2642918.2647398

[39] Xuyong Yang, Tao Mei, Ying-Qing Xu, Yong Rui, and Shipeng Li. 2016. Automatic generation of visual-textual presentation layout. *ACM Transactions on Multimedia Computing, Communications, and Applications* 12, 2 (2016), 1–22. https://doi.org/10.1145/2818709

[40] Xinru Zheng, Xiaotian Qiao, Ying Cao, and Rynson WH Lau. 2019. Content-aware generative modeling of graphic design layouts. *ACM Transactions on Graphics* 38, 4 (2019), 1–15. https://doi.org/10.1145/3306346.3322971