Karlsruhe Institute of Technology

Visualizing Dynamic Clustered Data Using Area-proportional Maps

Master Thesis of

Christian Schnorr

At the Department of Informatics
Institute of Theoretical Informatics, Algorithmics I

Reviewers:	Prof. Dr. Dorothea Wagner
	Prof. Dr. rer. nat. Peter Sanders
Advisor:	Dr. Tamara Mchedlidze

Time Period: November 14th, 2019 – June 16th, 2020

Acknowledgements

First of all, I would like to thank my advisor, Dr. Tamara Mchedlidze, for making this project possible through our weekly meetings and fruitful discussions, valuable feedback, and pointers in the right direction.

I am also grateful to the Algorithmics Group of Prof. Dr. Dorothea Wagner for the interesting courses over the last years, for giving me the opportunity to write this thesis, for their valuable insights early on in the project, and for the overall pleasant atmosphere.

Last but by no means least, I would like to thank my parents and grandparents and my good friend Johannes Hubert for their unfailing support and encouragement that kept me motivated on this final stretch, and for lending me some cores to help me crunch numbers for the evaluation of the framework discussed in this thesis.

Thank you!

Statement of Authorship

I hereby declare that this document has been composed by myself and describes my own work, unless otherwise acknowledged in the text.

Karlsruhe, June 11, 2020

Abstract

In this thesis, we explore how cluster information in graphs can be visualized naturally as countries on a map in a dynamic setting where the graph changes over time. We want there to be a strong correlation between a country's size and the size of the cluster it represents, and want to be able to react to dynamic changes of the graph in a way that preserves the viewer's mental model of the map. This is a challenging problem because popular algorithms for visualizing graphs as geographic-like maps struggle with clusters being fragmented across different countries on the map, and because redrawing the map for a new, albeit similar, input graph does generally not preserve the viewer's mental map.

To address these issues, we propose a framework that guarantees to keep clusters as continuous regions in the generated maps and supports dynamic inputs by allowing for small, incremental updates such as inserting or removing regions over time. We do this by working on an intermediate plane graph whose vertices correspond to clusters in the input graph, the cluster graph, and constructing the map as a contact representation of this cluster graph while accounting for the dynamic changes in both the cluster graph and its contact representation. This framework can be applied to a variety of real-world applications, allowing us to visualize clusters in the underlying data and how they change over time, thereby enabling viewers of the visualization to detect trends in the data easily.

Deutsche Zusammenfassung

In der vorliegenden Arbeit untersuchen wir, wie Clusterinformationen in Graphen als Länder einer Landkarte dargestellt werden können, wobei sich der Eingabegraph mit der Zeit verändert. Wichtig ist dabei eine starke Korrelation zwischen der Größe eines Landes auf der Karte und der Größe des entsprechenden Clusters. Außerdem wollen wir so auf Änderungen des Eingabegraphen reagieren können, dass ein Betrachter seine Orientierung auf der Karte nicht verliert. Das ist eine herausfordernde Problemstellung, da gängige Algorithmen zur landkartenähnlich Visualisierung von Graphen mit Clustern zu kämpfen haben, die über mehrere Länder auf der Karte fragmentiert sind. Außerdem geht die Orientierung des Betrachters bei einer Neuzeichnung der Karte für einen anderen, wennauch sehr ähnlichen Graphen, im Allgemeinen verloren.

Zur Lösung dieser Probleme schlagen wir ein Rahmenkonzept vor, welches garantiert, dass die Cluster jeweils zusammenhängende Regionen auf der Karte bilden. Außerdem werden dynamische Eingaben insofern unterstützt, als dass kleine, inkrementelle Änderungen wie z.B. das Einfügen oder Entfernen einer Region, durchgeführt werden können. Dafür arbeiten wir auf einem Hilfsgraphen, dessen Knoten den Clustern im Eingabegraph entsprechen, konstruieren die Karte als Kontaktrepräsentation dieses Graphen und passen bei Änderungen am Eingabegraphen sowohl den Hilfsgraphen als auch dessen Kontraktrepräsentation entsprechend an. Dieser Ansatz kann für eine Vielzahl von Anwendungen benutzt werden, um Cluster und deren zeitliche Enwicklung auf eine natürliche Art und Weise darzustellen und es dem Betrachter somit zu erleichtern, Trends in den zugrundeliegenden Daten zu erkennen.

Contents

1 Introduction **1**
1.1 Related Work 2
1.2 Contribution of this Thesis 5
1.3 Structure of this Thesis 5

2 Preliminaries **7**

3 Visualizing Static Input Graphs **11**
3.1 Transformation to Polygonal Dual 13
3.2 Drawing the Polygonal Dual 19

4 Visualizing Dynamic Input Graphs **23**
4.1 Inserting Vertices 26
4.2 Removing Vertices 29
4.3 Flipping Edges 31

5 Evaluation **35**
5.1 Quality Metrics 35
5.2 Test Case Generation 39
5.3 Implementation and Experiment Setup 41
5.4 Experimental Evaluation 42
5.5 Exemplary Maps 46

6 Conclusion **49**
6.1 Future Work 49

Bibliography **53**

1. Introduction

We live in a world with an abundance of data. Data for which it is no longer possible for humans alone to work through, or to make sense of, and data that is only going to increase in volume in the years to come. As the amount of data we work with soars to previously uncharted heights and analyzing it seems increasingly infeasible, visualizations retain their ability to help discover useful information, illustrate relationships in large sets of data, and efficiently communicate them with others.

Even in today's highly automated world, many processes still require some degree of human decision-making and interaction. Data visualizations are a great tool to inform these decisions as the visual perception channel has the highest information bandwidth of all human senses. In fact, we acquire more information through vision alone than through all other senses combined [War19]. However, as the mountains of data that we seek to understand keep growing, the human ability to process data remains mostly constant [Dac19]. Therefore, it becomes increasingly important to capture the most relevant aspects of the data and its trends with computer-aided visualizations.

In the fields of graph theory and graph drawing, for example, for larger graphs, global structures such as clusters and their relationships become increasingly important. In contrast, the importance of local features such as individual edges diminishes. Clustered data appears naturally in many real-world data sets. For example, one can group university students by their academic major, or politicians according to their party affiliation. When done correctly, clustering provides much additional structure to data and essentially functions as a high-level data aggregation that is easy to visualize while still conveying crucial global information. In this thesis, we propose a framework that explicitly visualizes clusters in the input graph as a high-level overview in addition to all the local details.

Given a map of the earth, it is remarkably easy to find out whether or not two countries are neighbors, or to compare their size. We are confronted with such tasks starting at a very young age, and absorbing data through this visual channel becomes very natural for most of us. In fact, we generally enjoy working with maps [SSK16]. Skupin and Fabrikant [SF03] realized the relevance of applying a cartographic perspective to general information visualization very early and motivated research of transferring the geographic map metaphor to non-geographic domains. In the following years, map-based graph visualizations, i.e., drawings of graphs in which the vertices' cluster information is explicitly encoded as colored regions of the plane, have been studied and evaluated in great detail. Such visualizations outperform traditional node-link diagrams on tasks related to detecting clustering information while not negatively affecting the performance regarding other tasks [SSKB14]. At the same time, they provide greater memorability of the data [SSKB15] and are more enjoyable to work with for many [SSK16]. In this thesis, we, too, shall utilize the map metaphor and encode the clustering information by placing all vertices of a cluster in a continuous region of the map.

In today's world, we are not only dealing with huge amounts of data. Data is also changing at a much greater pace than before: visualizations that were accurate yesterday may be outdated today. Many applications deal with volatile data by nature, such as stock prices or, in light of current events, the number of coronavirus infections. Therefore, it is crucial to not only visualize the data at a single point in time but to also visualize trends in the data as it evolves over time. However, visualizing a dynamic graph brings challenges of its own: We are essentially producing a sequence of snapshots at different points in time. Upon viewing such a snapshot, we inevitably create some internal representation of what we see in our minds. It is crucial for this mental model to remain consistent throughout the visualization: If it does not, viewers may not be able to see the overall trends in the dynamic data because they need re-orient often [BP90] [LLY06] [PHG06]. This fundamental aesthetic criterion of dynamic graph visualizations was coined *preserving the mental map* by Eades et al. [ELMS91] [MELS95] and is also known as *dynamic stability* [DG02]. In this thesis, we shall preserve the mental map of the dynamic map visualization by preserving the combinatorial embedding and outer face of the map and breaking changes down into small, incremental pieces that can be applied to the dynamic map easily.

Cartograms are maps in which geographic regions appear skewed such that their areas are proportional to some statistic, e.g., the population or gross domestic product of a country. Although cartograms are traditionally used to visualize demographic data and are based on real geographic maps, extensive studies related to human perception [NK16] [NAK18] give us a few valuable insights that translate to our problem definition. First, the area is a strong visual variable that can be interpreted naturally. Second, having a before/after-comparison allows viewers to detect trends in the underlying data easily. In this thesis, we therefore aim to create maps whose regions' areas are proportional to their respective cluster sizes and adopt established quality metrics of cartograms for the visualizations we produce.

A real-world application with great potential to benefit from this framework and its goals is the visualization of opinion networks [BHMvS19] and how they evolve over time in particular. An opinion network is represented as a weighted graph whose vertices are so-called opinion vectors and whose edge weights represent the similarity between two opinion vectors. Betz et al. [BHMvS19] cluster the vertices to group similar opinions together, visualize the graph as a map in which each cluster corresponds to a country, and draw the original graph on top of this map following the GMap algorithm [GHK09]. Naturally, such an opinion network is dynamic in a sense where existing opinions can change and new opinions can be incorporated over time: clusters can grow or shrink, appear or disappear, merge or split, etc. We are interested in visualizing such processes effectively to find and communicate important trends in the underlying data.

Before diving into the contributions of this thesis, we discuss related work that we build upon to better understand the aspects we are going to address.

1.1 Related Work

GMap and OpMap

GMap [GHK09] is a framework for visualizing graphs as geographical maps. The first step consists of embedding the input graph in the plane and performing a cluster analysis using arbitrary user-provided algorithms. Embedding and clustering are typically done independently from one another; however, the algorithms should be well-matched such that clusters form continuous regions in the computed embedding. A combination of force-directed graph drawing and modularity-based clustering methods is commonly used. GMap then creates a map by computing the Voronoi diagram of the graph's vertices as per the computed embedding and merging adjacent cells whose vertices belong to the same cluster. Eventually, the original graph's edges are drawn on top of this map.

In a follow-up paper, Mashima et al. [MKH11] built upon GMap to visualize dynamic input graphs. To maintain the viewer's mental map, they create a "canonical map" storing positional information of a much larger graph than the one that is eventually shown to the viewer. At different points in time, only the most prominent vertices and clusters of the canonical map are then visualized. Because the canonical map is much less volatile, vertices and clusters do not move as much on the resulting map.

The algorithm for visualizing opinion networks by Betz et al. mentioned earlier is called OpMap. OpMap focuses on clustering and weighting the opinion vectors in the network and delegates the visualization to GMap. However, the clustering is performed on the abstract input graph, independent from the force-directed embedding algorithm. This can be problematic, because if the clustering and the computed embedding do not fit together well, one ends up with many fragmented, noncontinuous clusters on the map. OpMap has only been evaluated for relatively small input sizes and works with static inputs only. We want to explore an approach that supports larger, and most importantly, dynamic, opinion networks.

Therefore, in this thesis, we propose a framework that clusters the graph and extracts relevant features first, and only then embeds the graph in the plane while keeping the clusters connected.

Cartograms

Cartograms have been studied for more than 50 years [Tob04], and there are lots of fundamentally different approaches to generate different kinds of cartograms [NK16]. Their quality is generally assessed by a combination of statistical accuracy, topological accuracy, and geographic accuracy [AKV15]: *Statistical accuracy* describes how closely the areas of the modified geographic regions match the variable of interest, *topological accuracy* to what degree the original region adjacencies are preserved, and *geographic accuracy* how well the shapes and positions of the distorted regions resemble their original. Geographic accuracy is closely related to the preservation of the mental map: it captures a viewer's ability to intuitively recognize the regions and therefore detect the trends in the underlying data. Although traditionally used for other applications, cartograms are closely related to our problem statement, and the quality metrics mentioned above translate directly to our use case. While we do not have a reference map for the initial area-proportional map like cartograms do, we can use that initial map as a reference map that we can base geographical accuracy on when incorporating dynamic updates of the input graph. There is a large body of literature on the generation of cartograms [Tob04] [AKV15] [NK16]. Here, we mention only the most relevant techniques and explain to what degree we have adopted them.

Gastner and Newman [GN04] propose a physical model based on diffusion to generate cartograms: They rasterize the original map into a two-dimensional matrix with the values being the initial densities, i.e., the statistical values divided by the regions' areas at any given point. This matrix is then used to precompute the gradient of the diffusion field and the pathlines of these "density particles" as they diffuse through the map and equalize the overall density. The pathlines essentially map locations on the original map to their location in the diffused map and can be used to draw the distorted, density-equalizing map. However, due to the rasterization and heavy precomputation of pathlines, this algorithm is not well-suited for our dynamic setting in which densities can change and are not necessarily known a priori.

Kämper et al. [KKN13] start with a polygonal map and transform every edge into a circular arc that can bend to realize the desired areas of individual regions. They use a max-flow-based formulation on the map's dual graph to determine how the area should be distributed among the regions and solve for the circular arc radii. However, the degree to which the edges can bend is heavily restricted since the circular arcs may not touch or cross, making it difficult for circular arc cartograms to achieve good statistical accuracy. The resulting regions' shapes also appear very artificial, unlike those found on a real geographic map.

Alam et al. [ABF$^+$13b] show how air-pressure-based models for the general floorplan problem such as [ITK98] and [Fel13] can be applied to generating rectilinear cartograms. They give a force-directed heuristic to compute the cartogram iteratively and experimentally show very fast convergence to more than 99% accuracy. Each region is assigned a target area based on the statistic one wants to visualize. They then compute the pressure in each region based on its current area and target area and use it to grow or shrink the regions iteratively by shifting its boundaries. This heuristic motivates the force-directed formulation of a pivotal part of our framework whose theoretical background we discuss in the following paragraph.

Force-directed Graph Drawing

Force-directed graph drawing algorithms regard the graph to be visualized as a physical system in which the vertices are individual particles, and several forces are acting on said particles. These forces are defined such that they act to bring the system into a stable equilibrium position in which its potential energy is at a local minimum, and the resulting drawing has certain desired features. Eades [Ead84] first used a combination of attractive forces between adjacent vertices based on physical springs and repulsive forces between all pairs of vertices based on electric repulsion. These forces result in adjacent vertices being pulled together while non-adjacent vertices are pushed further apart. The drawings resulting from force-directed algorithms are generally visually appealing and easy to grasp [Kob13].

Bertault [Ber99] designed a force-directed algorithm called PrEd that imposes additional constraints on the displacement of vertices. In PrEd vertices are only allowed to move in a way that preserves the edge crossing properties of the initial layout, i.e., existing edge crossings are preserved, and no new edge crossings are introduced. Simonetto et al. [SAAB11] introduced an improved version of this algorithm called ImPrEd, which is more flexible and performs much better on larger input graphs.

In this thesis, we use a force-directed algorithm with custom forces to control the layout of our polygonal maps. The forces we use are based on the aforementioned "pressure" in the map's regions. We must pay attention not to introduce edge crossings when tweaking the map's layout because regions are not allowed to overlap on geographical maps.

Area Universality

Area-universal graphs are plane graphs that can realize any area assignment to its internal faces with straight-line edges. Research on area-universality gives us important theoretical bounds on the statistical accuracy we can achieve with polygonal maps.

Back in 1992, Thomassen [Tho92] showed that plane cubic graphs are area-universal. For polygonal maps, this means that, as long as at most three regions meet in a point, we can achieve perfect statistical accuracy for arbitrary region weights. This is because if we were to eliminate all corners that are part of only two regions' boundaries, all remaining corners would be are part of precisely three regions' boundaries. Therefore, one can interpret the map as a plane cubic graph.

Kleist [Kle18] [Kle19] showed that the 1-subdivision of any plane graph is area-universal. With just one bend per edge, any plane graph can therefore be drawn with arbitrary prescribed face areas. This property translates to polygonal maps, too. As long as there exists a point that's part of at most two regions' boundaries between two points that are part of more than two regions' boundaries, we can lift the requirement of no more than three regions meeting in a point. Most importantly, we can relax the requirements while still being able to achieve perfect statistical accuracy for arbitrary region weights.

1.2 Contribution of this Thesis

The main problem with GMap is the independent nature of the clustering and embedding phases: If clustering is done first, a force-directed embedding algorithm likely causes a high degree of fragmented and noncontinuous regions [MKH11]. OpMap manages to somewhat diminish this effect with a smart choice of forces that makes use of the clustering information [vS17]. Embedding the graph first and then clustering it using a geometric clustering algorithm is dangerous, too, because this may detect clusters that do not exist in the abstract data. Most force-directed algorithms used with GMap also lack the ability to specify the desired sizes and shapes of the clusters.

OpMap, as implemented in [vS17], supports only some degree of dynamicity: while new vertices can be added to existing clusters dynamically, the set of clusters is determined statically. This means that as new opinions are incorporated into the opinion network, no new clusters can form.

In this thesis, we address the aforementioned issues in related work and try to overcome them. In particular, we design an algorithm that constructs maps in which the countries are guaranteed to be continuous, whose areas are close to proportional to the respective cluster sizes, and whose shapes are somewhat organic to resemble real-world maps. The algorithm also allows for dynamic updates such as inserting new clusters, removing existing clusters, tweaking cluster adjacencies, or changing cluster weights. To evaluate our algorithm, we adopt several measures from literature, capturing both the map's accuracy and the quality of its regions' shapes. Finally, we experimentally evaluate the quality of the maps created by our algorithm using randomly generated data sets.

1.3 Structure of this Thesis

In Chapter 2, we present preliminary definitions of various concepts that we base our algorithms and discussions on, some of which were already brought up earlier in this chapter without formal definitions. We then define our algorithmic pipeline for static inputs in Chapter 3 and adapt it to dynamic inputs in Chapter 4, discussing possible implementations of the pipeline's phases along the way. In Chapter 5, we discuss different quantifiable measures to assess the quality of the visualizations produced by our implementation and apply those in an experimental evaluation on a large number of test inputs. Eventually, in Chapter 6, we summarize our framework and its shortcomings and explore how future research can improve upon it.

2. Preliminaries

When dealing with sets, we use the operator $+$ in place of $\cup$ if we want to emphasize that the operands are disjoint. Similarly, we use the operator $-$ in place of $\setminus$ to stress that the right operand is a subset of the left operand. We sometimes use set elements where sets would be mathematically correct, e.g., $\{1, 2\} + 3 = \{1, 2, 3\}$. In these cases, the element is wrapped in a set implicitly.

Graphs

A *(simple, undirected) graph* G is a tuple $G = (V, E)$ of *vertices* V and *edges* $E \subseteq \{\{u, v\} | (u, v) \in V^2\}$. For an arbitrary graph H, we sometimes denote its vertices by $V(H)$ and its edges by $E(H)$. Graphs can be weighted, assigning real-valued weights to its vertices or edges. We differentiate between vertex-weighted graphs that provide a weight function $w_v \colon V \to \mathbb{R}$ and edge-weighted graphs that provide a weight function $w_e \colon E \to \mathbb{R}$. When graphs are weighted, we typically include the weight functions in the graph tuple, e.g., $G_w = (V, E, w_v)$. Of course, graphs can be both vertex- and edge-weighted.

Given a graph $G = (V, E)$, two vertices $u, v \in V$ are said to be *adjacent* if there's an edge connecting u to v, i.e., $\{u, v\} \in E$. Adjacent vertices are also called *neighbors*. The *degree* $\deg(v)$ of a vertex v is its number of adjacent vertices, i.e., $\deg(v) \coloneqq |\{u | \{u, v\} \in E\}|$. An edge $\{u, v\} \in E$ is said to be *incident* to its *endpoints* u and v.

A *path* p is a (non-empty) sequence of edges that connects a sequence of distinct vertices $(v_1, \ldots, v_n)$, i.e., $p = (\{v_1, v_2\}, \ldots, \{v_{n-1}, v_n\}) \wedge \forall i \neq j \colon v_i \neq v_j$. The first and last vertices v_1, v_n of a path are called its *endpoints*. A path p's *length* is the number of edges in the path.

A graph $G = (V, E)$ is *connected* if there exists a path between any pair of vertices in the graph. G is said to be *k-(vertex)-connected* if one needs to remove k or more vertices and their incident edges to disconnect G. Graphs that are 2-vertex-connected are also called *biconnected*.

A *subgraph* H of a graph G, denoted $H \subseteq G$, is a graph with a subset of vertices and edges from G, i.e., $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. A subgraph is said to be *spanning* if it includes all vertices of the original graph, i.e., $V(H) = V(G)$.

Graph Clustering

Given a graph $G = (V, E)$, a *clustering* $\mathcal{C}$ of G is a partition $\mathcal{C} = \{C_1, \dots, C_k\}$ of its vertices V into non-empty, pairwise disjoint *clusters* of vertices, i.e., $\bigcup_{i=1}^{k} C_i = V$, $\forall i \colon C_i \neq \varnothing$, and $\forall i \neq j \colon C_i \cap C_j = \varnothing$.

A clustering $\mathcal{C} = \{C_1, \dots, C_k\}$ of a graph G induces a so-called *cluster graph* of G, a vertex- and edge-weighted graph $G_{\mathcal{C}} = (V_{\mathcal{C}}, E_{\mathcal{C}}, w_v, w_e)$ whose vertices are the clusters C_i:

$$\begin{aligned} V_{\mathcal{C}} &\coloneqq \{C_1, \dots, C_k\} \\ E_{\mathcal{C}} &\coloneqq \{\{C_i, C_j\} | (C_i, C_j) \in \mathcal{C}^2, C_i \neq C_j\} \\ w_v(C_i) &\mapsto |C_i| \\ w_e(\{C_i, C_j\}) &\mapsto |\{(u, v) \in E | u \in C_i \wedge v \in C_j\}| \end{aligned}$$

Graph Operations

Given a graph $G = (V, E)$, one can *subdivide* an edge $e \coloneqq \{u, v\} \in E$ by removing the edge e and inserting a new vertex w with edges to both u and v. Subdividing the edge $\{u, v\}$ therefore produces the graph $G' = (V', E')$ with $V' = V + w$ and $E' = E - \{u, v\} + \{u, w\} + \{w, v\}$. A graph H is said to be a *subdivision* of G if H can be obtained from G by repeatedly subdividing edges. H is a *1-subdivision* of G if it can be obtained from G by subdividing every edge exactly once.

One can also *contract* an edge $\{u, v\}$ of a graph $G = (V, E)$ if its endpoints u and v do not have any neighbors in common, deleting the edge and "merging" its endpoints into a new vertex w. Doing so removes the edges $E_- \coloneqq \{\{x, y\} \in E | u \in \{x, y\} \vee v \in \{x, y\}\}$ and creates new edges $E_+ \coloneqq \{\{w, x\} | \{u, x\} \in E \vee \{v, x\} \in E\}$, producing a new graph $G' = (V', E')$ with $V' = V - u - v + w$ and $E' = E - E_- + E_+$. Similarly, a vertex v of degree 2 can be *smoothed*, replacing it and its incident edges with a new edge $\{u, w\}$ between its original neighbors u and v, i.e., $G' = (G', E')$ with $V' = V - v$ and $E' = E - \{u, v\} - \{v, w\} + \{u, w\}$.

Graph Drawings

A *drawing* Γ of a graph $G = (V, E)$ maps its vertices to distinct points in the plane and its edges to continuous curves between its endpoints in the plane. The graph G is called *planar* if it permits a drawing in which its edges do not self-intersect and do not cross other edges, i.e., the graph's edges only intersect in shared endpoints. Such a drawing is called a *planar drawing* of G. If all edges of G are line segments, we also call it a *planar straight-line drawing.*

A planar drawing Γ of a graph G partitions the plane into mutually disjoint regions called *faces*. All faces but one, the *internal faces*, are bounded; the unbounded face is called the *outer face*. Faces are identified by the cyclic order of the vertices that are encountered when walking along its boundary [ADK+15] in such a way that the inside of the face lies on the left side of the boundary. Vertices and edges are said to be *incident* to a face if they lie on the face's boundary. Similarly, two faces are said to be *adjacent* if their boundaries have an edge in common. If vertices or edges lie on the outer face's boundary, we call them *external*; otherwise, we call them *internal.*

A *planar embedding* ϕ of G is an equivalence class of planar drawings that define the same set of faces for G. Therefore, a planar embedding ϕ uniquely determines the internal faces and the outer face, or, equivalently, the cyclic order of the edges incident to each vertex and the choice of the outer face [ADK+15].

We also use the term *plane graph* to refer to a planar graph G in combination with a planar embedding ϕ and denote it as G_ϕ. A plane graph G can be *face-weighted* if there's a weight function $w_f \colon F_{\text{int}}(G) \to \mathbb{R}$ assigning real numbers to its internal faces $F_{\text{int}}(G)$.

A planar graph G is said to be *maximal* or *triangulated* if adding any edge to G would result in a nonplanar graph. Thus, in a maximal plane graph, all faces, including the outer face, are triangles. Similarly, a plane graph G_ϕ is said to be *internally triangulated* if all internal faces are triangles.

A *multigraph* is a graph that, as opposed to a simple graph, can have multiple edges between a pair of vertices as well as so-called loops, i.e., edges connecting a vertex to itself. The *dual (graph)* G^*_ϕ of a plane graph G_ϕ is a plane multigraph consisting of a vertex v_f for every face f of G_ϕ and edges connecting the vertices corresponding to the incident faces of each edge in the graph G_ϕ. The dual is embedded in such a way that the cyclic order of the edges incident to each vertex is equivalent to the cyclic order of the edges bounding the respective faces in the original graph. In relation to the dual G^*_ϕ, the original graph G_ϕ is also called the *primal (graph)*. The *weak dual* G^-_ϕ of a plane graph G_ϕ is its dual without the vertex corresponding to the G_ϕ's outer face [FGH74].

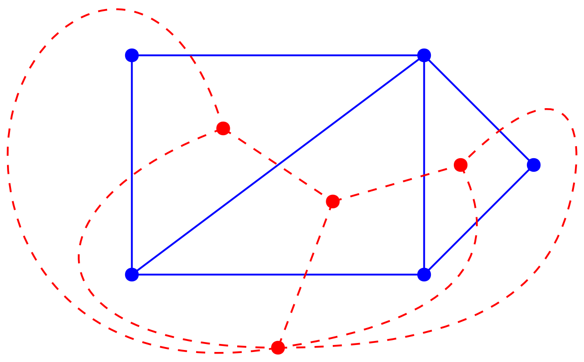


Figure 2.1: A plane graph G (in blue) and its dual G^* (in red).

Figure 2.1 illustrates an example of a plane graph and its dual. Forming the dual of a plane graph essentially turns vertices into faces and vice versa and "rotates" the edges of the plane graph, as illustrated in Figure 2.1. Note that the planar embedding of the primal graph uniquely determines the embedding of its dual. When forming the dual of the dual of a plane graph, we get back the original plane graph, i.e., $(G^*)^* = G$.

Contact Representations

In a *(simple) contact representation* of a plane graph G_ϕ, the vertices are represented by pairwise internally disjoint regions of the plane, and edges are implicitly represented by the non-trivial contacts between the corresponding regions [ABF$^+$13a]. Because G_ϕ is simple, the regions corresponding to adjacent vertices in G_ϕ must only share a single continuous boundary in the contact representation. A contact representation is said to be *hole-free* if all regions other than the unbounded outer region have a corresponding vertex in G_ϕ.

Unless otherwise noted, we assume that contact representations are hole-free and that they respect the original graph G_ϕ's embedding; i.e., for all regions, the cyclic order of the boundaries with adjacent regions is equivalent to the cyclic order of the incident edges of the corresponding vertex in G_ϕ to its respective neighbors.

Given a plane graph G_ϕ, a contact representation of G_ϕ with axis-aligned rectilinear polygons is called a *rectilinear dual* of G_ϕ [ABF$^+$13b]. The name "dual" is fitting here because vertices of the original graph turn into faces, or regions, of the rectilinear dual. In fact, we can view the rectilinear dual as a plane graph itself: the polygons' corners are vertices connected by edges as determined by the polygons' sides, as illustrated in

Figure 2.2. We use the term rectilinear dual to refer to the actual arrangement of rectilinear polygons and the induced plane graph interchangeably.

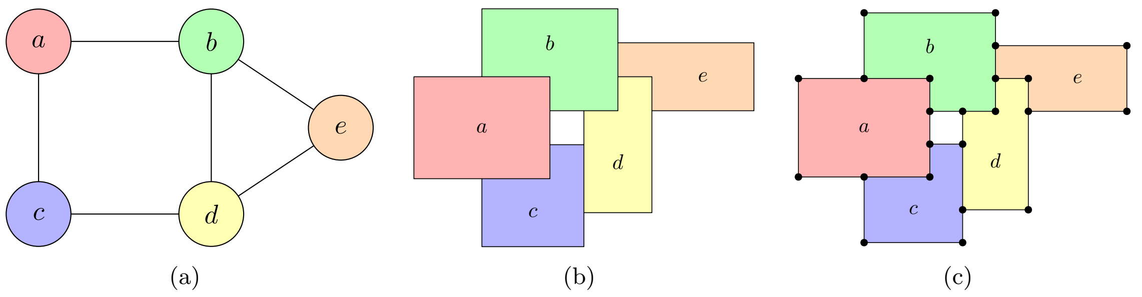


Figure 2.2: A plane graph G_ϕ (a) and a possible rectilinear dual of G_ϕ with a hole, as a plain arrangement of polygons (b), and construed as yet another plane graph (c).

A contact representation can be *weighted*, assigning a real weight to all regions corresponding to vertices of its primal. We define the *normalized cartographic error* [AKV15] of a weighted contact representation as

$$\max_{v \in V(G)} \frac{|A'(v) - w(v)|}{\max\{A'(v), w(v)\}},$$

where $w(v)$ is the prescribed area of the region corresponding to vertex v and $A'(v)$ is its actual area $A(v)$, normalized such that constant factors drop out, i.e.,

$$A'(v) := A(v) \cdot \frac{\sum\limits_{u \in V} w(u)}{\sum\limits_{u \in V} A(u)}.$$

We say that a weighted contact representation is *ε-area-proportional* if its normalized cartographic error is less than or equal to ε.

Area Universality

A plane graph G with internal faces F_{int} is *area universal* if, for every area assignment of its internal faces $A\colon F_{\text{int}} \to \mathbb{R}_+$, there exists a planar straight-line drawing of G with the same embedding such that all internal faces $f \in F_{\text{int}}$ have the area $A(f)$ prescribed by A.

Recall that Thomassen [Tho92] showed that all (planar embeddings of) cubic graphs, i.e., graphs in which all vertices have degree 3, are area-universal, and that Kleist [Kle19] showed that all plane 1-subdivisions of planar graphs are area-universal, too.

3. Visualizing Static Input Graphs

In this chapter, we provide a detailed description of our problem statement, formalize it, and discuss our solution in the case of static input graphs.

Our goal is to visualize clusters of a larger graph, such as an opinion network, as an artificial map. In this map, each cluster of the larger graph is represented by a continuous region whose area is approximately proportional to the cluster's size, and neighboring regions indicate similarities between the respective clusters.

Unlike OpMap and GMap, which have similar goals, clustering the larger graph is not part of the framework we discuss here. Instead, we assume the larger graph is clustered externally before being fed into our framework, producing some cluster graph. The first step of our framework takes a spanning subgraph of this cluster graph as input and builds an initial contact representation in which all regions are simple polygons. Such a contact representation does not exist for arbitrary spanning subgraphs of the cluster graph; for example, the subgraph must be planar. We formalize the additional requirements we impose on the spanning subgraph in a bit. The contact representation is then tweaked, displacing the polygons' corners in such a way that their areas are approximately proportional to the clusters' weights, and that they have somewhat organic shapes. To formalize this, we require two short definitions:

Definition 3.1. A contact representation in which all regions are simple polygons is called a *polygonal contact representation.*

Definition 3.2. Given a plane graph G, a polygonal contact representation of G is called a *polygonal dual* of G.

Note that polygonal duals are a generalization of rectilinear duals and that, analogous to rectilinear duals, a polygonal dual can be interpreted as a plane graph itself: the polygons' corners translate to vertices in this graph, and their sides translate to edges.

With these definitions in place, we can formalize the structure of our framework: Our input is a biconnected and internally triangulated plane subgraph of a cluster graph that we call the *filtered cluster graph* $G^{\mathcal{C}}$. We start by forming an initial weighted polygonal dual of $G^{\mathcal{C}}$, the *initial map* Γ^*. Γ^* inherits its (face) weights from the vertices of $G^{\mathcal{C}}$. Then, we turn Γ^* into an ε-area-proportional, polygonal contact representation of $G^{\mathcal{C}}$ for some small $\varepsilon > 0$, the *ε-proportional map* $\Gamma^{\propto}$ of $G^{\mathcal{C}}$. We do this by displacing the contact representation

Γ^*'s vertices while preserving its planarity. We implement this step using a force-directed algorithm.

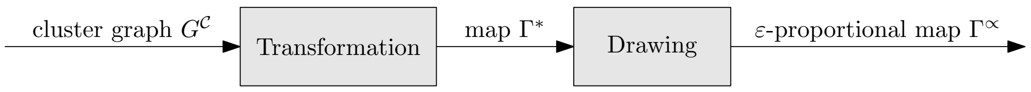

cluster graph $G^{\mathcal{C}}$
Transformation
map Γ^*
Drawing
ε-proportional map $\Gamma^{\propto}$

Figure 3.1: Overview of the algorithmic pipeline for static input graphs.

Let us break down the requirements we impose on the filtered cluster graph $G^{\mathcal{C}}$:

- $G^{\mathcal{C}}$ is given as a plane graph. Of course, $G^{\mathcal{C}}$ must be planar such that there exists a contact representation of $G^{\mathcal{C}}$. We require a specific planar embedding of $G^{\mathcal{C}}$ such that the "arrangement" of the clusters, or of the regions in the eventual map, is predetermined. This requirement becomes essential in Chapter 4, where we start incorporating dynamic updates into the filtered cluster graph and the resulting maps.
- $G^{\mathcal{C}}$ must be internally triangulated such that the maps Γ^* and $\Gamma^{\propto}$ are hole-free. Applying the map metaphor, this translates to there not being any lakes or rivers separating countries on the map. Figure 2.2 illustrates how internal faces on four or more vertices creates holes in contact representations.
- $G^{\mathcal{C}}$ must be biconnected such that, in combination with the internal triangulatedness, no vertex appears on the outer face multiple times. The region of such a vertex in the polygonal dual would need to have multiple boundaries with the outer face, which we specifically excluded in Chapter 2.
- $G^{\mathcal{C}}$ must be vertex-weighted such that the map Γ^* can inherit its vertex weights. We require these weights to determine the areas the polygonal regions are supposed to have in $\Gamma^{\propto}$.

Many real-world applications, such as visualizing opinion networks, do not produce a filtered cluster graph directly. In order for our framework to be applicable, one may need to prepend a clustering phase that turns an arbitrary input graph G into a biconnected and internally triangulated plane subgraph $G^{\mathcal{C}}$ of a cluster graph of G that can then be processed by our framework:

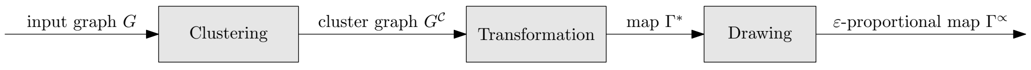

input graph G
Clustering
cluster graph $G^{\mathcal{C}}$
Transformation
map Γ^*
Drawing
ε-proportional map $\Gamma^{\propto}$

Figure 3.2: Overview of a possible algorithmic pipeline for generic applications.

We will now discuss our implementation of the transformation and drawing phases of the pipeline in detail.

3.1 Transformation to Polygonal Dual

In this step of the pipeline, we take a filtered cluster graph $G^{\mathcal{C}}$ and form a polygonal dual of thereof, resulting in the map Γ^*. To formalize the relationship between these two graphs, we require the concept of the augmented dual:

Definition 3.3. The *augmented dual* G^+ of a plane graph G is the plane multigraph obtained by first placing a new vertex v^+ in the outer face of G, connecting it to all vertices on the outer face, in order, without introducing edge crossings, and then forming its dual.

Figure 3.3 illustrates the formation of the augmented dual G^+ of a plane graph G (Figure 3.3(a)). We add the helper vertex v^+ and helper edges $\{v^+, \cdot\}$ in the outer face in Figure 3.3(b). We draw the helper edges as in [Wag16] because it does not matter where this helper vertex lies in the plane — all that matters is that each pair of adjacent vertices on the outer face forms a new triangular face with v^+. In Figure 3.3(c), we overlay the dual vertices and edges in red. Figure 3.3(d) shows just the augmented dual of G.

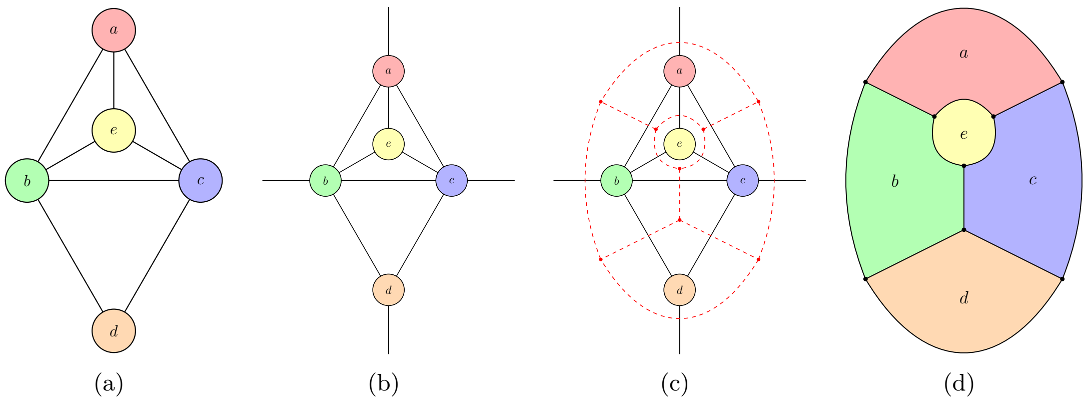


Figure 3.3: Step-by-step representation of forming a plane graph's augmented dual.

Note that analogous to the regular dual, the weak dual of the augmented dual of a plane graph G is G again, i.e., $(G^+)^- = G$.

The augmented dual G^+ (Figure 3.3(d)) of a biconnected and internally triangulated plane graph G (Figure 3.3(a)) essentially is a contact representation thereof. In a polygonal dual, however, the edges cannot be curves and must be polylines instead. Therefore, one can interpret the map Γ^* as a subdivision of the augmented dual $(G^{\mathcal{C}})^+$ in combination with a planar straight-line drawing thereof.

Algorithm Overview

The underlying idea of creating the initial map Γ^* is as follows: Given a filtered cluster graph $G^{\mathcal{C}}$, we place vertices on edges of the outer face of $G^{\mathcal{C}}$ (these edges bound additional triangular faces after adding the helper vertex in the process of forming the augmented dual) and inside the internal faces of $G^{\mathcal{C}}$. We then connect these vertices if their corresponding faces are adjacent. Connecting two vertices may require additional subdivision vertices or bends in order not to introduce edge crossings — we use a single bend per edge.

The following figure illustrates an example:

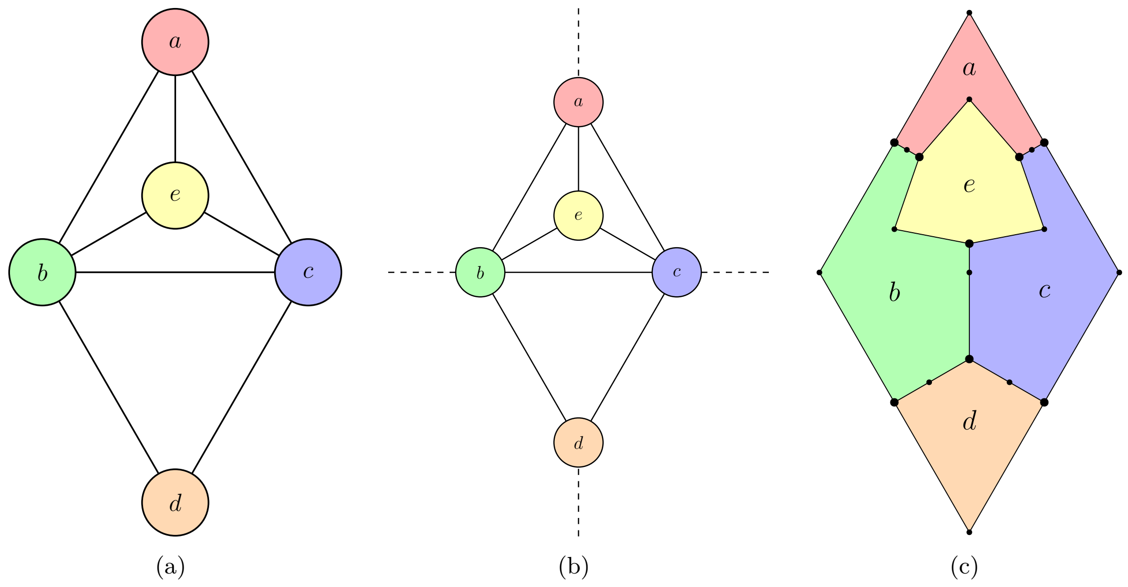


Figure 3.4: A filtered cluster graph $G^{\mathcal{C}}$ (a), the helper graph $G^{\mathcal{C}}_{+}$ (b), and the polygonal dual Γ^{*} of $G^{\mathcal{C}}$ as produced by Algorithm 3.1 (c).

Before feeding the filtered cluster graph $G^{\mathcal{C}}$ into the algorithm, we construct a planar straight-line drawing $\Gamma^{\mathcal{C}}$ of $G^{\mathcal{C}}$ with the same planar embedding. According to Fáry's theorem [Fá48] [Wag36] [Ste51], such a drawing exists for every plane graph. There are numerous popular algorithms to construct such a drawing, e.g., Tutte's method [Tut63], the shift method [DFPP90], or the Schnyder realizer method [Sch90].

Algorithm Implementation

Algorithm 3.1: Transformation to Polygonal Dual

```
Input: filtered cluster graph $G^{\mathcal{C}}$ and a planar straight-line drawing $\Gamma^{\mathcal{C}}$ thereof
Output: polygonal dual $\Gamma^*$ of $G^{\mathcal{C}}$, in the form of a 1-subdivision of the augmented
        dual of $G^{\mathcal{C}}$ and planar straight-line drawing thereof
1  create empty contact representation $\Gamma^*$
   // compute dual vertices and edges
2  foreach internal face $f$ in $G^{\mathcal{C}}$ do
3      add “internal face vertex” $v_f$ to $\Gamma^*$
4      position $v_f$ at barycenter of $f$ in $G^{\mathcal{C}}$
5  foreach edge $\{u, v\}$ in $G^{\mathcal{C}}$ do
6      if $\{u, v\}$ is incident to two different internal faces $f, g$ in $G^{\mathcal{C}}$ then
7          add “subdivision vertex” $v_{\text{sub}}$ to $\Gamma^*$
8          position $v_{\text{sub}}$ at midpoint of $\{u, v\}$ in $G^{\mathcal{C}}$
9          add edge between $v_f$ and $v_{\text{sub}}$ to $\Gamma^*$
10         add edge between $v_{\text{sub}}$ and $v_g$ to $\Gamma^*$
11     else if $\{u, v\}$ is incident to a single internal face $f$ in $G^{\mathcal{C}}$ then
12         add “outer edge vertex” $v_{\{u,v\}}$ to $G_{\text{init}}$
13         position $v_{\{u,v\}}$ at midpoint of $\{u, v\}$ in $G^{\mathcal{C}}$
14         add “subdivision vertex” $v_{\text{sub}}$ to $\Gamma^*$
15         position $v_{\text{sub}}$ at midpoint of segment between barycenter of $f$ and midpoint
             of $\{u, v\}$ in $G^{\mathcal{C}}$
16         add edge between $v_{\{u,v\}}$ and $v_{\text{sub}}$ to $\Gamma^*$
17         add edge between $v_{\text{sub}}$ and $v_f$ to $\Gamma^*$
18 foreach incident edges $\{\{u, v\}, \{v, w\}\}$ on outer face of $G^{\mathcal{C}}$ do
19     add “subdivision vertex” $v_{\text{sub}}$ to $\Gamma^*$
20     position $v_{\text{sub}}$ at position of $v$ in $G^{\mathcal{C}}$
21     add edge between $v_{\{u,v\}}$ and $v_{\text{sub}}$ to $\Gamma^*$
22     add edge between $v_{\text{sub}}$ and $v_{\{v,w\}}$ to $\Gamma^*$
   // compute faces in $\Gamma^*$ and match them to vertices in $G^{\mathcal{C}}$
23 foreach vertex $u$ in $G^{\mathcal{C}}$ do
24     endpoints $\leftarrow$ ()
25     foreach adjacent pair $(v, w)$ of neighbors of $u$ in counterclockwise order do
26         if $(u, v, w)$ bound a triangular face $f$ in counterclockwise order in $G^{\mathcal{C}}$ then
27             append $v_f$ to endpoints
28         else
29             append $v_{\{u,v\}}$ to endpoints
30             append $v_{\{u,w\}}$ to endpoints
31     foreach adjacent pair $(v, w)$ in endpoints do
32         insert subdivision vertex connecting $v$ to $w$ between $v$ and $w$ in endpoints
33     define $f_u$ as internal face on endpoints in $\Gamma^*$
34     set weight of $f_u$ in $\Gamma^*$ to weight of $u$ in $G^{\mathcal{C}}$
35 return $\Gamma^*$
```

Algorithm Correctness

We show the algorithm's correctness in two steps. First, we show that the algorithm does indeed construct a 1-subdivision of the augmented dual of $G^{\mathcal{C}}$. Second, we show that the constructed drawing thereof is planar, making it a contact representation of $G^{\mathcal{C}}$ as previously explained and illustrated in Figure 3.3.

Recall that $(G^{\mathcal{C}})^+$ is a contact representation of $G^{\mathcal{C}}$. We can construct $(G^{\mathcal{C}})^+$ via a helper graph $G^{\mathcal{C}}_+$ that we obtain by inserting a helper vertex v^+ in the outer face of $G^{\mathcal{C}}$ and connecting it to all vertices on the outer face of $G^{\mathcal{C}}$, in order, without introducing edge crossings. The contact representation $(G^{\mathcal{C}})^+$ then is the regular dual of $G^{\mathcal{C}}_+$, i.e., $(G^{\mathcal{C}})^+ = (G^{\mathcal{C}}_+)^*$. By adding the helper vertex v^+ and helper edges $\{v^+, \cdot\}$, the edges on the outer face of $G^{\mathcal{C}}$ bound new triangular faces of $G^{\mathcal{C}}_+$.

The dual of a plane graph has vertices in each of the primal graph's faces. We place these vertices in the original internal faces in line 3 and in — or rather on — the new triangular faces in $G^{\mathcal{C}}_+$ in line 12. Vertices in the dual are adjacent if their respective faces are adjacent in the primal. As a result, we require a dual edge in $(G^{\mathcal{C}}_+)^*$ for every edge separating two faces in the primal $G^{\mathcal{C}}_+$. We add these edges in lines 9 to 10 (for adjacent internal faces in $G^{\mathcal{C}}$), in lines 16 to 17 (for edges separating an internal face in $G^{\mathcal{C}}$ from a new face in $G^{\mathcal{C}}_+$), and in lines 21 to 22 (for helper edges $\{v^+, \cdot\}$ separating two new faces in $G^{\mathcal{C}}_+$). Because we place all bends of the dual edges on the respective primal edge, the cyclic order of the edges incident to each vertex is equivalent to the cyclic order of the edges bounding the respective faces in the primal.

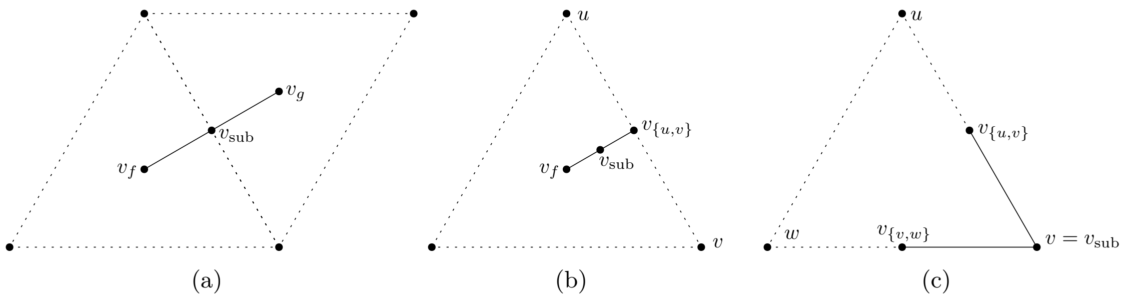


Figure 3.5: Construction of the dual edges as per Algorithm 3.1 between two internal faces in $G^{\mathcal{C}}$ (a), between an internal face of $G^{\mathcal{C}}$ and a new face in $G^{\mathcal{C}}_+$ (b), and between two new faces in $G^{\mathcal{C}}_+$ (c). The dotted lines are edges in $G^{\mathcal{C}}$.

It is left to show that the placement of vertices and edges produces a crossing-free drawing. To do so, we consider the edges from non-subdivision vertices in $(G^{\mathcal{C}}_+)^*$ to their neighbors. Recall that those vertices correspond to faces in $G^{\mathcal{C}}_+$. Following the construction illustrated in Figures 3.5(a) to 3.5(b), the three incident edges to a vertex corresponding to an internal face of $G^{\mathcal{C}}$ lie completely inside said face. Because these edges go straight towards the midpoint of the incident edges, they do not self-intersect.

The vertices corresponding to new faces in $G^{\mathcal{C}}_+$ lie on the midpoint of the bounding edge that already exists in $G^{\mathcal{C}}$. We construct the dual edges connecting two of those vertices such that they partition the boundary of the outer face of $G^{\mathcal{C}}$ (see Figure 3.5(c)) and therefore do not intersect any of the edges discussed so far. Their edge to the vertex corresponding to the adjacent internal face goes straight towards said vertex, meeting it halfway, as illustrated in Figure 3.5(b). Because some v_f's incident edges all go straight towards its bounding edges as outlined above, this kind of edge does not introduce any edge crossings either. As a result, the constructed graph and drawing is a 1-subdivision of the augmented dual of $G^{\mathcal{C}}$, and the drawing is planar.

Once the algorithm has constructed Γ^*, we compute an explicit representation of its internal faces and the corresponding vertices in the primal. We do so for the augmented dual $(G^{\mathcal{C}})^+$ first (loop in line 25) and then insert the subdivision vertices (loop in line 31). The face in $(G^{\mathcal{C}})^+$ that corresponds to a vertex v of $G^{\mathcal{C}}$ is bounded by vertices corresponding to internal faces of $G^{\mathcal{C}}$ that contain v, plus the vertices corresponding to the additional triangular faces in $G^{\mathcal{C}}_+$ in case v lies on the outer face of $G^{\mathcal{C}}$. Iterating over adjacent pairs (e_1, e_2) of incident edges in counterclockwise order correctly detects the internal faces between e_1 and e_2 and the case where e_1 and e_2 wrap around on the outside in order. The orientation check in line 26 is required to prevent the wraparound case from being incorrectly classified as an internal face, as would be the case for the vertex $v := d$ and edges $e_1 := \{d, b\}, e_2 := \{d, c\}$ in Figure 3.4.

Algorithm Runtime

To compute the input graph's faces, we replace every edge with two inversely oriented, directed edges. We then repeatedly pick any unmarked edge and form a directed cycle by following the next outgoing edge according to the embedding, marking the edges as we go. Once all edges have been marked, we have found all faces. The outer face is the single face whose interior is on the left when traveling along its bounding edges. We therefore find the graph's faces in $\Theta(n+m)$, where $n = |V(G^{\mathcal{C}})|$ and $m = |E(G^{\mathcal{C}})|$.

The input graph has $\Theta(n)$ internal faces. They are all triangles; therefore, we can compute their barycenter in $\Theta(1)$ each (line 4, line 15). By keeping track of which faces an edge of the input graph is incident to while computing the faces as outlined above, we allow for $\Theta(1)$ lookups in line 6 and line 11. As a result, the loop in lines 2 to 4 runs in $\Theta(n)$, the loop in lines 5 to 17 in $\Theta(m)$, and the loop in lines 18 to 22 in $\Theta(m)$.

The loop in lines 23 to 34 processes every vertex once in line 23 and every edge twice in line 25. We can check if the vertices u, v, w form a triangle in constant time (line 26) by checking if there is an edge between v and w. Considering each of the $\Theta(n+m)$ vertices of the generated graph appears in `endpoints` in no more than two iterations of the loop in lines 23 to 34 and all those vertices have degree 3, we can find their shared neighbor in constant time and implement the entire loop to run in $\Theta(n+m)$.

Therefore, the entire algorithm can be implemented to run in $\Theta(n+m)$.

Theoretical Bounds

Do we have to subdivide edges of the augmented dual of our filtered cluster graph $G^{\mathcal{C}}$ to get a valid polygonal dual of $G^{\mathcal{C}}$? Although the augmented dual $(G^{\mathcal{C}})^+$ is plane by definition, it is not immediately evident that there exists a planar straight-line drawing of $(G^{\mathcal{C}})^+$ — and that is what we need for it to be a polygonal contact representation.

In our case, however, $(G^{\mathcal{C}})^+$ is simple; i.e., there are no loops or multiple adjacencies. Recall that $G^{\mathcal{C}}$ is biconnected and internally triangulated. As a consequence, adding the helper vertex in the outer face of $G^{\mathcal{C}}$ and connecting it to all vertices on the outer face creates a fully triangulated, simple graph. In a simple triangulated graph, there are no two edges that are incident to the same faces (only those would create multiple adjacencies when forming the dual) and no edges that have the same face on both sides (only those would create loops when forming the dual) either. The augmented dual $(G^{\mathcal{C}})^+$ is therefore simple, and, according to Fáry's theorem [Fá48], there exists a planar straight-line drawing of $(G^{\mathcal{C}})^+$ respecting its original planar embedding.

In addition to having a planar straight-line drawing, $(G^{\mathcal{C}})^+$ is also a cubic graph, i.e., one in which all vertices have degree 3. This is because $G^{\mathcal{C}}$ with the helper vertex is a triangulated graph, meaning every face is incident to exactly three edges, which turns into every vertex being incident to exactly three edges when forming the dual. Therefore, according to the results of Thomassen [Tho92], $(G^{\mathcal{C}})^+$ is area-universal. This result implies that regardless of the actual face weights that $G^{\mathcal{C}}$ prescribes, there exists a planar straight-line drawing of $(G^{\mathcal{C}})^+$ realizing those weights/areas.

As a result, even without subdividing edges in $(G^{\mathcal{C}})^+$, we could choose the map Γ^* in such a way that it already has perfect statistical accuracy. Such a drawing of $(G^{\mathcal{C}})^+$ is non-trivial to compute, though, and creates undesired region shapes according to our other quality metrics. Instead, the algorithm we proposed here subdivides all edges of the augmented dual $(G^{\mathcal{C}})^+$ once and leaves us with a decent initial drawing and enough degrees of freedom to optimize for other quality metrics in Section 3.2.

3.2 Drawing the Polygonal Dual

In the second step of the pipeline, we apply a force-directed graph drawing algorithm to the initial map Γ^* produced by Algorithm 3.1 to generate an ε-proportional map $\Gamma^{\propto}$ for some small $\varepsilon > 0$.

In a force-directed algorithm, we interpret a graph's vertices as particles in a physical system. Based on the structure of the graph and the relative position of the particles, we define several forces that act to bring the system to a stable equilibrium position in which its potential energy is at a local minimum. In these equilibrium positions, the system is in a somewhat relaxed state that, in the context of graph drawing, generally is a visually appealing drawing of the graph. We find an equilibrium position by iteratively computing the net force acting on each particle and displacing it by a small amount in the direction of the net force, based on its absolute value.

Forces

We define the forces that exist in our particle system with two goals in mind: First, the faces of the graph (the regions of the map) to have an area ought to be close to proportional to some prescribed value. Second, we want the faces to be *locally fat*. Our intuitive understanding of local fatness is that a region should not have drawn-out, tight corridors. We formalize and discuss different quantifiable measures that aim to capture the local fatness of regions in Chapter 5. Note that both of these are soft requirements. Planarity of the resulting drawing, on the other hand, is a hard requirement that we must preserve at all costs.

Let us now discuss the concrete force components that act to bring our particle system to an equilibrium position.

- **Air Pressure:** Motivated by Alam et al. [ABF$^+$13b], we treat the polygonal regions as volumes of some amount of air equal to the respective face's weight. This idea allows us to define an analog to air pressure in the polygonal regions that exert forces on the regions' edges. This force is responsible for growing faces that are currently compressed and shrinking faces that are currently larger than they should be, therefore working towards the statistical accuracy of the generated map.

  We want this force and the overall drawing to be agnostic to constant factors of the face weights and therefore compute the normalized pressure $P(f)$ in an internal face f as
  $$P(f) \coloneqq \frac{w(f)}{A(f)} \cdot \frac{\sum_{g \in F} A(g)}{\sum_{g \in F} w(g)},$$
  where $A(f)$ is the area currently covered by some internal face f and $w(f)$ its weight. We set the normalized pressure in the outer face to the weighted average normalized pressure, i.e.,
  $$P(f_{\text{outer}}) \coloneqq \frac{\sum_{f \in F} A(f) \cdot P(f)}{\sum_{f \in F} A(f)} = 1.$$

  Physical pressure is the ratio of force to area. Accordingly, the air pressure in each of the regions f exerts a force on each bounding edge $e = \{u, v\}$ based on the pressure's magnitude and the edge's length $l(e)$ in relation to the entire region's boundary's length $l(f)$. We orient $e = \{u, v\}$ such that u directly precedes v on the boundary of f, and define the force as
  $$\vec{F}_P((u, v); f) \coloneqq 3 \cdot P(f) \cdot \frac{l(e)}{l(f)} \cdot \mathrm{Norm}(\mathrm{Perp}(\overrightarrow{vu})), \tag{3.1}$$

where Perp(·) rotates a vector by 90° in counterclockwise direction, and Norm(·) normalizes a vector to unit length. We apply $\vec{F}_P(\{u,v\}; f)$ to both endpoints u and v of the edge, as illustrated in Figure 3.6.

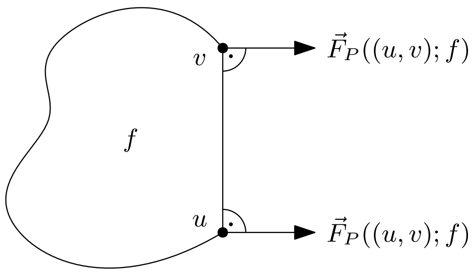


Figure 3.6: Forces exerted on the endpoints of the edge $e = \{u, v\}$ by the air pressure in the region f.

Considering every edge is incident to exactly two faces, we can write the net force exerted on an oriented edge $e = (u, v)$ incident to face f on the left and face g on the right as

$$\vec{F}_P((u,v); f, g) \coloneqq 3 \cdot \left(P(g) \cdot \frac{l(e)}{l(g)} - P(f) \cdot \frac{l(e)}{l(f)} \right) \cdot \operatorname{Norm}(\operatorname{Perp}(\overrightarrow{uv})),$$

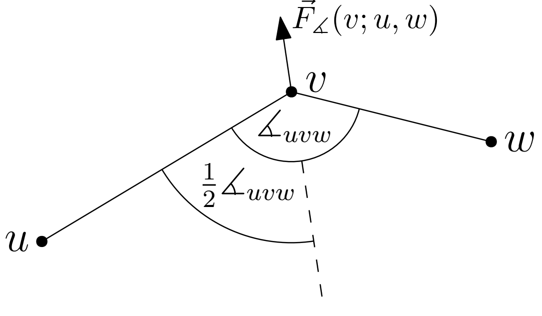
matching the force that Alam et al. [ABF+13b] use for computing their cartograms.

- **Angular Resolution:** Internal face angles close to 0° cause tight corridors in the form of pointy spikes and angles close to 360° the opposite — both features we want to avoid if possible. Accordingly, we define a force that optimizes angular resolution, i.e., a force that tries to evenly distribute the angles formed by the incident edges around a vertex v at $\frac{360°}{\deg(v)}$ each.

  Let v be a vertex and let u and w be two successive neighbors of u in counterclockwise order. Also, let $\measuredangle_{uvw} \in [0°, 360°)$ denote the normalized angle from u via v to w measured in counterclockwise direction. With this, we define the force $\vec{F}_\measuredangle(v; u, w)$ as

$$\vec{F}_\measuredangle(v; u, w) \coloneqq \frac{1}{2} \cdot \frac{\frac{360°}{\deg(u)} - \measuredangle_{uvw}}{\measuredangle_{uvw}} \cdot \operatorname{Bsc}(\angle_{uvw}). \tag{3.2}$$

  Here, $\operatorname{Bsc}(\angle_{uvw})$ computes the normalized bisector of the given angle, i.e., $\operatorname{Norm}(\overrightarrow{vu})$ rotated by $\frac{1}{2}\measuredangle_{uvw}$ in counterclockwise direction. Figure 3.7 illustrates the construction of $\vec{F}_\measuredangle(v; u, w)$.



$\vec{F}_\measuredangle(v; u, w)$
v
$\measuredangle_{uvw}$
w
$\frac{1}{2}\measuredangle_{uvw}$
u

Figure 3.7: Force exerted on some vertex v with successive neighbors u and w of v, where $\measuredangle_{uvw} > \frac{360°}{\deg(v)}$.

We apply $\vec{F}_{\measuredangle}(v; u, w)$ to v for all triplets (u, v, w) of vertices v and successive neighbors u and w of v. If the angle at v is currently too small, the force acts to move v along the bisector, thereby increasing the angle; otherwise, it acts to move v against the bisector, thereby decreasing the angle.

Argyriou et al. [ABS13] use a similar force to obtain uniform angles around all vertices. However, instead of applying the force to the vertex v whose angular resolution we want to improve as we do, they apply perpendicular forces to its neighbors. In our tests, the approach of Argyriou et al. has shown not to converge well due to us not having a force acting towards a uniform edge length, as discussed below in a bit.

- **Vertex-vertex repulsion:** We define a repulsive force between pairs of vertices to prevent the vertices from clumping together. Preventing this is important because, to preserve planarity, vertices that are very close to others will need to have their movement restricted severely, possibly hindering us from satisfying our aesthetic criteria. We think of the vertices as charged particles that push each other away and define a repulsive force based on Coulomb's law. This force is exerted along the lines connecting pairs of vertices and its magnitude depends on the Euclidean distance of the points:

$$\vec{F}_{\leftrightarrow}(u; v) := 25 \cdot \frac{1}{\|\overrightarrow{uv}\|^2} \cdot \operatorname{Norm}(\overrightarrow{uv}) \tag{3.3}$$

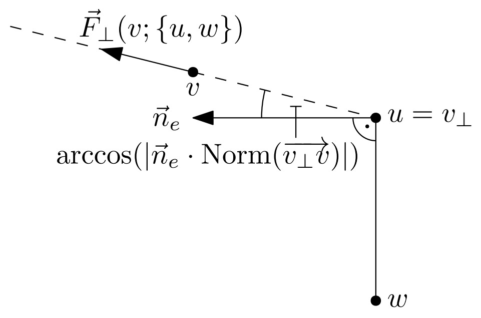
For pairs $(u, v) \in V^2, u \neq v$, we apply $\vec{F}_{\leftrightarrow}(u; v)$ to u. We restrict ourselves to pairs (u, v) where u and v lie together on the boundary of some face for performance reasons, and because for other pairs, there would be other vertices or edges between u and v that would push the two vertices apart.

This kind of force was first used by Eades [Ead84], albeit only for non-adjacent pairs of vertices. We apply this force to adjacent vertices, too, because we do not use spring-like forces between adjacent vertices trying to achieve a uniform edge length (and therefore some non-zero distance) as Eades does, as discussed below.

- **Vertex-edge repulsion:** In another attempt to prevent tight corridors from forming, we define an additional repulsive force between vertices and edges. Given an edge $e = \{u, w\}$ and non-incident vertex v, we define $v_\perp$ as the point on the segment from u to w with the smallest Euclidean distance to v. We then define the repulsive force as

$$\vec{F}_{\perp}(v; \{u, w\}) := 10 \cdot \frac{1}{\|\overrightarrow{v_\perp v}\|^2} \cdot |\vec{n}_e \cdot \operatorname{Norm}(\overrightarrow{v_\perp v})| \cdot \operatorname{Norm}(\overrightarrow{v_\perp v}), \tag{3.4}$$

where $\vec{n}_e$ is the normalized normal vector of the edge e pointing in either direction. The following figure illustrates the construction of the force $\vec{F}_{\perp}(v; \{u, w\})$:



$\vec{F}_{\perp}(v; \{u, w\})$
v
$\vec{n}_e$
$u = v_\perp$
$\arccos(|\vec{n}_e \cdot \operatorname{Norm}(\overrightarrow{v_\perp v})|)$
w

Figure 3.8: Force exerted on some vertex v by a non-incident edge $\{u, w\}$.

Analogous to the vertex-vertex-repulsion discussed above, we compute the force $\vec{F}_{\perp}(v;\{u,w\})$ only for vertices and edges that lie together on the boundary of some face and apply it to v.

This repulsive force is very similar to what Bertault uses in PrEd [Ber99]. However, we have an additional dot product term for how v is positioned relative to the edge $\{u,w\}$, whereas Bertault drops the force altogether if the orthogonal projection of v onto the line through u and w does not lie between u and w. In our tests, Bertault's definition resulted in very unstable simulations as these force swayed back and forth between having a full impact and no impact at all.

The constant factors of the force components above were determined experimentally and yielded good results for a variety of randomly generated graphs.

Note that we do not define attractive forces between adjacent vertices as most traditional force-directed algorithms do. Such an attractive force is generally used to keep adjacent vertices close together while non-adjacent vertices are pushed further apart by the repulsive forces. Many graph drawing algorithms for which uniform edge length is one of the desired features in equilibrium use such a force. In our case, however, there is no correlation between the extent of a region and the number of edges on its boundary and, subsequently, the length of the edges on its boundary. Including such an attractive force here would be counterproductive.

In our implementation, we also smooth vertices of degree 2 that become too close to other vertices and subdivide edges that become too long in order to create more degrees of freedom in the respective face's shape. To do so, we first compute the average edge length $\bar{l}$ as

$$\bar{l} := \frac{1}{|E|} \cdot \sum_{e \in E} l(e).$$

Then, at each iteration, we subdivide edges that are longer than $2\bar{l}$ at their midpoint and smooth vertices whose distance to the closest other vertex is $\frac{1}{10}\bar{l}$ or less if we can do so without introducing edge crossings.

Preserving Edge Crossing and Combinatorial Properties

When iteratively displacing the map's vertices according to the forces defined above, we must pay close attention to not accidentally introduce edge crossings or change the cyclic order of incident edges around the vertices.

To do so, we adopt the rules of ImPrEd [SAAB11] that ensure no edge crossings are created. At each step of the algorithm, ImPrEd computes the maximum distance each of the vertices is allowed to move such that the drawing's edge crossing properties are guaranteed to be preserved. The maximum displacement per vertex is computed in eight general directions, zones of 45° each. The actual displacement of the vertices is then clamped at the maximum distance the vertex can safely move in the desired direction.

As a byproduct, these rules ensure that the cyclic order of incident edges around the vertices stays the same, thereby preserving the map's combinatorial properties.

4. Visualizing Dynamic Input Graphs

Extending the approach discussed in the previous section to dynamic input graphs is challenging because we must try to preserve the viewer's mental map as the underlying data changes over time. We want the visualization at different points in time to be similar enough so that the viewer can easily tell what parts have changed [MKH11], yet allow for the required changes in geography and topology. Still, changes between visualizations at consecutive points in time should minimize movement and allow for smooth animations therebetween.

Extending the pipeline for static inputs discussed in the previous section in a straightforward way will not satisfy these requirements. Running through the entire pipeline with a different, albeit similar, input graph, may result in a completely different visualization, thereby destroying the viewer's mental map. This is because even though the combinatorial arrangement of the regions of the map $\Gamma_t^\propto$ is predetermined by the planar embedding of the filtered cluster graph $G_t^\mathcal{C}$, the independent nature of the runs through the pipeline can result in regions with drastically different shapes and (absolute) positions. Therefore, we extend the pipeline in a way that allows for small, incremental changes to be propagated through the pipeline and to eventually be applied to the previous output in a way that preserves the viewer's mental map.

We extend the pipeline for static input by an incremental transformation phase. This phase takes two inputs: A map Γ_t^* that the pipeline previously produced as output for some filtered cluster graph $G_t^\mathcal{C}$, and a sequence of operations on said cluster graph, that, when applied to $G_t^\mathcal{C}$, yields the cluster graph $G_{t+1}^\mathcal{C}$. The incremental transformation phase then determines how these operations translate to a polygonal dual of $G_t^\mathcal{C}$ and applies the translated operations to Γ_t^*, producing Γ_{t+1}^*, a polygonal dual of $G_{t+1}^\mathcal{C}$. We then feed this new polygonal dual back into the drawing phase to make it an ε-area-proportional contact representation $\Gamma_{t+1}^\propto$ for some small $\varepsilon > 0$ and to improve the local fatness of the map's regions.

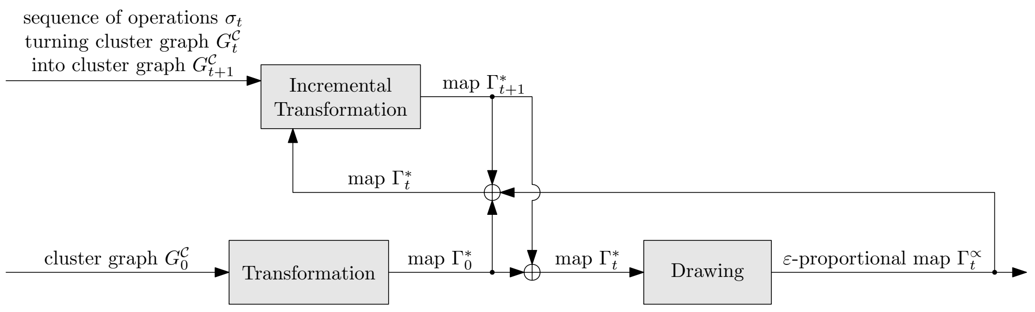

sequence of operations σ_t turning cluster graph $G_t^{\mathcal{C}}$ into cluster graph $G_{t+1}^{\mathcal{C}}$
Incremental Transformation
map Γ_{t+1}^*
map Γ_t^*
cluster graph $G_0^{\mathcal{C}}$
Transformation
map Γ_0^*
map Γ_t^*
Drawing
ε-proportional map $\Gamma_t^{\propto}$

Figure 4.1: Overview of the algorithmic pipeline for dynamic input graphs.

Real-world applications, such as visualizing a dynamic opinion network, need a way to feed a sequence of operations on the filtered cluster graph into our framework. One could compute such a sequence of operations σ_t by prepending an incremental clustering phase that translates changes of the simple input graph G_t into changes of its filtered cluster graph $G_t^{\mathcal{C}}$. However, such a sequence of operations σ_t is only meaningful in combination with a graph to which these operations can be applied. One must consequently provide the previously-produced cluster graph $G_t^{\mathcal{C}}$ as an additional input to the incremental clustering phase such that it can tailor its output to the cluster graph that has already been locked in in an earlier run through the pipeline. The following figure illustrates this possible extension to our pipeline:

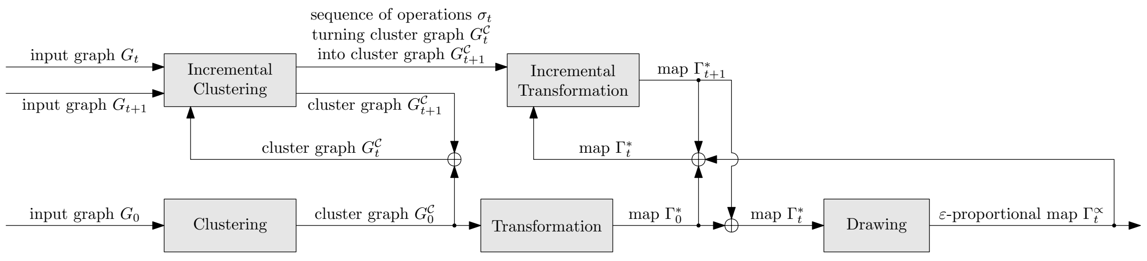

input graph G_t
input graph G_{t+1}
Incremental Clustering
sequence of operations σ_t turning cluster graph $G_t^{\mathcal{C}}$ into cluster graph $G_{t+1}^{\mathcal{C}}$
cluster graph $G_{t+1}^{\mathcal{C}}$
cluster graph $G_t^{\mathcal{C}}$
Incremental Transformation
map Γ_{t+1}^*
map Γ_t^*
input graph G_0
Clustering
cluster graph $G_0^{\mathcal{C}}$
Transformation
map Γ_0^*
map Γ_t^*
Drawing
ε-proportional map $\Gamma_t^{\propto}$

Figure 4.2: Overview of a possible algorithmic pipeline for generic applications.

Extending the pipeline to allow the propagation of small, incremental changes of the input graph has numerous benefits other than preserving the viewer's mental map:

- It allows highly efficient implementations of the incremental parts of the pipeline as only the aspects that have actually changed in the input graph or intermediate products need to be processed and propagated further along the pipeline.
- It makes the pipeline highly parallelizable for dynamic inputs: when a later phase is processing changes, an earlier phase can already start processing new changes independently. With our force-directed implementation of the drawing phase, we can even incorporate dynamic updates while the drawing phase is still running, even if it has not converged yet: We pause the force simulation, feed the current map Γ_t^* into the incremental transformation phase to incorporate the dynamic updates, and then resume the simulation with the updated map Γ_{t+1}^* produced by the incremental transformation phase.
- It efficiently supports dynamic input in an online setting, i.e., a setting in which the incremental changes are not known in advance, such as when visualizing live data.

Supported Operations

Our pipeline supports numerous classes of atomic operations on the filtered cluster graph, such as inserting and removing vertices and edges, flipping edges, or simply changing a cluster's weight. By composing multiple atomic operations in a sequence, more drastic changes can be made to the filtered cluster graph. Nonetheless, the operations are applied one after the other in our pipeline.

The most straightforward operation of all is changing a vertex v's weight: We simply take the previous ε-proportional map $\Gamma_t^{\propto}$, update the weight of the face f_v corresponding to the vertex v, and declare that as the new map Γ_{t+1}^{*}. The map Γ_{t+1}^{*} then runs through the drawing phase again to account for the updated face weights.

Implementing the remaining operations as part of the incremental transformation is a little more challenging, and we will discuss those in great detail in the following sections.

4.1 Inserting Vertices

When a new cluster appears in our underlying data set, we want to add a new vertex to the cluster graph and have that change translate to an existing polygonal dual thereof. We distinguish between adding a new vertex in an internal face and adding a new vertex in the outer face because slightly different rules apply.

Inserting Vertices Inside

All internal faces of the filtered cluster graph $G_t^{\mathcal{C}}$ are triangles. If we add a vertex in one of the triangular faces, we must also add edges to the three vertices bounding the face without introducing edge crossings to preserve the graph's internal triangulatedness. Figure 4.3 shows an example of a valid vertex insertion into an internal face.

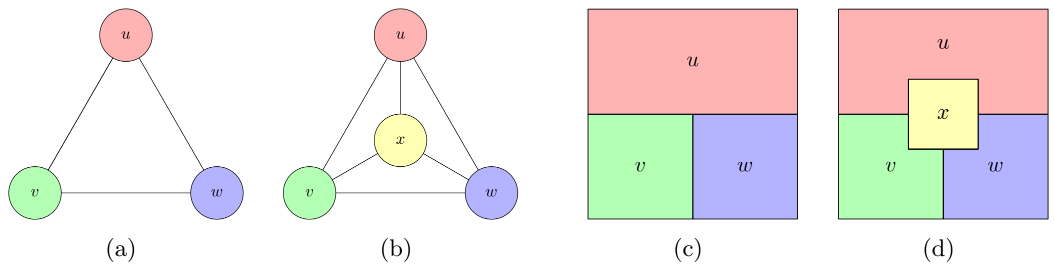


Figure 4.3: A cluster graph and a polygonal dual thereof, before (a, c) and after (b, d) inserting the vertex x in the triangular face uvw.

Let u, v, and w be the vertices bounding an internal face of the cluster graph in counter-clockwise order and let x be the new vertex we want to add inside said face.

Let p_{uvw} be the vertex of the map Γ_t^* that is common to faces u, v, and w. Also, let p_{uv}, p_{vw}, and p_{wu} denote the subdivision vertices on the boundary of these faces that are incident to p_{uwv}, as illustrated in Figure 4.4(a). If one of the boundaries consist of only one edge, we subdivide the edge at its midpoint first.

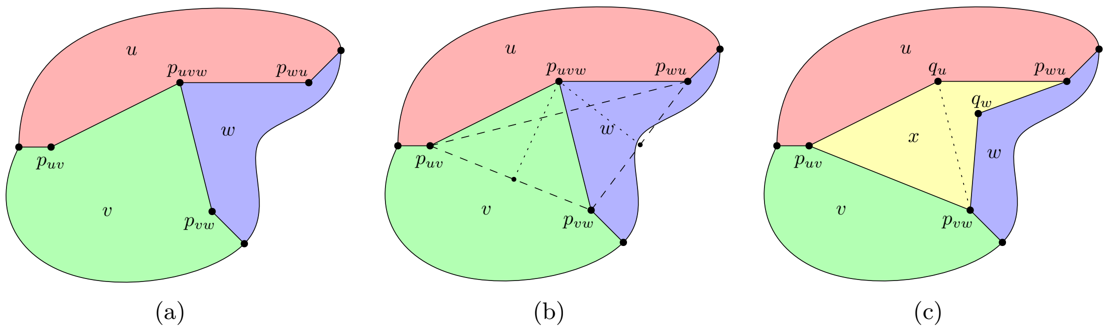


Figure 4.4: Inserting an internal face x at the point where three faces u, v, w meet.

We then want to remove the vertex p_{uvw} along with its incident edges and insert edges between the subdivision vertices instead to bound a new face for x, as illustrated by the dashed lines in Figure 4.4(b). Doing so naïvely may introduce edge crossings, though, and we generally need to add bends in the form of subdivision vertices to those edges.

To determine if we need a bend and where to position it, we distinguish three cases, all of which are illustrated in Figure 4.4(c). Let (a, b, c) be a rotation of (u, v, w), i.e., $(a, b, c) \in \{(u, v, w), (v, w, u), (w, u, v)\}$. Then the three cases are the following:

- If we can place an edge between p_{ab} and p_{bc} without introducing an edge crossing, we do not need to bend the edge (face v in Figure 4.4(c)).
- Otherwise, if the internal angle of face a at p_{abc}, $\angle_{p_{ab}p_{abc}p_{bc}}$, is 180° or more, we bend the edge using a subdivision vertex q_a at p_{abc}. (face u in Figure 4.4(c))
- Otherwise, we do a binary search for a bend location in the form of a subdivision vertex q_a on the outward-pointing bisector of the angle $\angle_{p_{ab}p_{abc}p_{bc}}$ (dotted lines in Figure 4.4(b)). We start at the point where the bisector intersects the line segment from p_{ab} to p_{bc} and keep dividing the remaining distance to p_{abc} in half until we find a bend location for which the bent edge from p_{ab} to p_{bc} would not introduce edge crossings. (face w in Figure 4.4(c))

Considering at most one of the internal angles at p_{uvw} can be 180° or more, we place at most one subdivision vertex there. By removing the vertex p_{uvw} and inserting the edges $\{p_{uv}, p_{vw}\}$, $\{p_{vw}, p_{wu}\}$, and $\{p_{wu}, p_{uv}\}$, potentially with bends q_v, q_w, and q_u, we create an internal face for x without introducing edge crossings.

Inserting Vertices Outside

Alternatively, we can add a new vertex in the outer face of the filtered cluster graph. Such a vertex must be connected to at least two vertices on the outer face to preserve the graph's biconnectivity. Its neighbors must also form a path on the original boundary of the cluster graph in order not to create holes and thereby violate its internal triangulatedness.

We restrict ourselves to adding new vertices in the outer face that are made incident to exactly two neighboring vertices here. Additional edges can be inserted separately afterward, as discussed in Section 4.3. Let $\{u, v\}$ be an edge on the outer face. Then we support adding a new vertex x in the outer face and connecting it to both u and v. Figure 4.5 shows an example of a valid vertex insertion into the outer face.

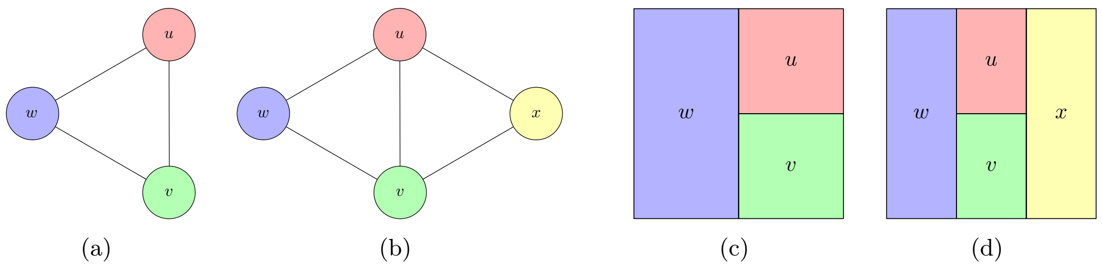


Figure 4.5: A cluster graph and a polygonal dual thereof, before (a, c) and after (b, d) inserting the vertex x on the outer face and connecting it to u and v.

This operation is a special case of the vertex insertion discussed above: In the polygonal dual, instead of creating a new face between three internal faces, we need to create a new face between two internal faces and the implicit outer face.

Recall that the polygonal dual is a subdivision of the augmented dual $(G_t^{\mathcal{C}})^+$ of the cluster graph $G_t^{\mathcal{C}}$. In the construction of the augmented dual from Definition 3.3, we insert a helper vertex incident to all vertices on the outer face of the primal graph and then form the regular dual. That is why the polygonal dual has an internal face for each of the vertices in $(G_t^{\mathcal{C}})^+$. The polygonal dual also has an implicit outer face — an ocean in the context of maps — that corresponds to the helper vertex inserted during the construction of the augmented dual, as illustrated in Figure 4.6.

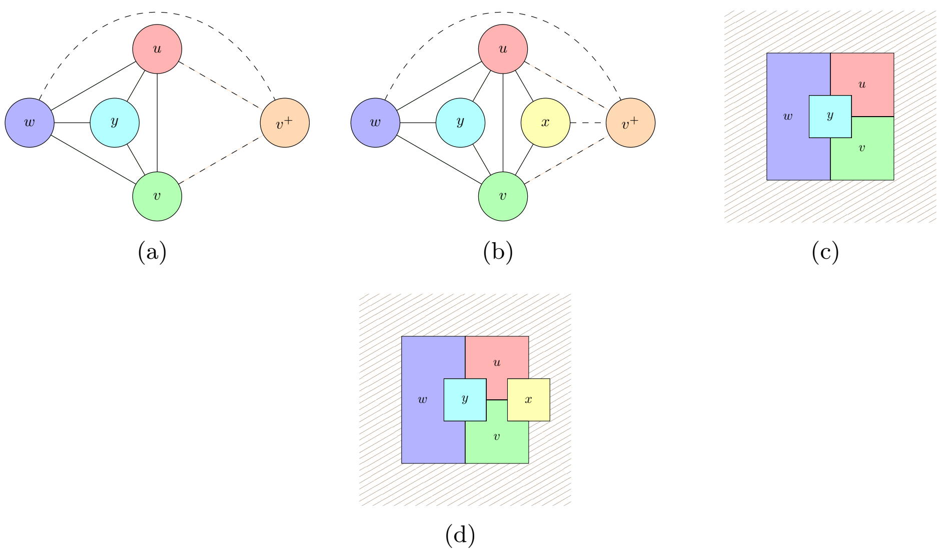


Figure 4.6: A cluster graph and a polygonal dual thereof with explicit helper vertex v^+ and outer face, before (a, c) and after (b, d) inserting the vertex x in the triangle u and v form with v^+.

Also recall that the cluster graph with this helper vertex is triangulated. Two arbitrary adjacent vertices on the outer face of $G_t^{\mathcal{C}}$ form a triangular face with the helper vertex v^+. As a result, inserting a new vertex x in the outer face as described above is equivalent to inserting x in the triangular face u and v form with the helper vertex v^+, or, in the dual, inserting a face at the point where the faces u and v meet the implicit outer face.

The construction outlined above translates 1-to-1 to inserting a vertex in the outer face, except one of the three neighboring faces is the implicit outer face.

4.2 Removing Vertices

Clusters in the underlying data set can also fade and eventually cease to exist. In this case, we want to be able to remove existing vertices of the cluster graph. We make the same distinction here as when inserting vertices: we can remove internal vertices or vertices that lie on the cluster graph's outer face.

Removing Internal Vertices

When removing an internal vertex of the filtered cluster filter, we must ensure that the graph remains internally triangulated. This property is preserved iff the internal vertex we want to remove has degree 3. Removing vertices with a higher degree would create holes in the graph. If one wants to remove a vertex with degree 4 or higher, one must first change the adjacencies on the inside of the cluster graph using edge flips, discussed in Section 4.3. Figure 4.7 shows an example of a valid removal of an internal vertex.

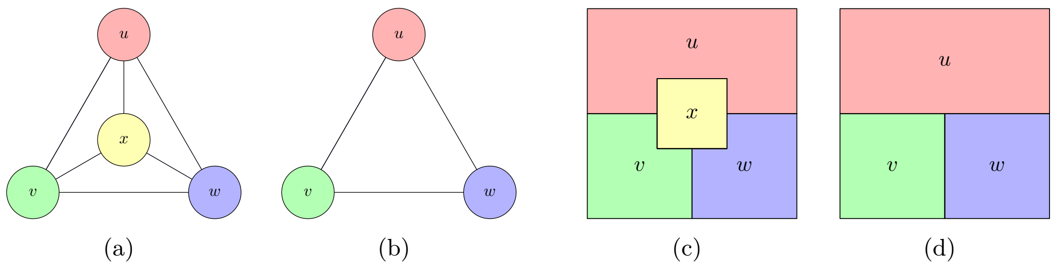


Figure 4.7: A cluster graph and a polygonal dual thereof, before (a, c) and after (d, d) removing the internal vertex x with degree 3.

In order not to create holes in the contact representation, the incident faces must take over the area that the face to be removed used to occupy. Let x denote the internal vertex we want to remove in the primal and u, v, and w its three neighbors. Let p_{uvx} (p_{vwx}, p_{uwx}) denote the vertex where face x meets faces u and v (v and w, w and u) in the dual. Figure 4.8(a) shows how these vertices might look for the example from above.

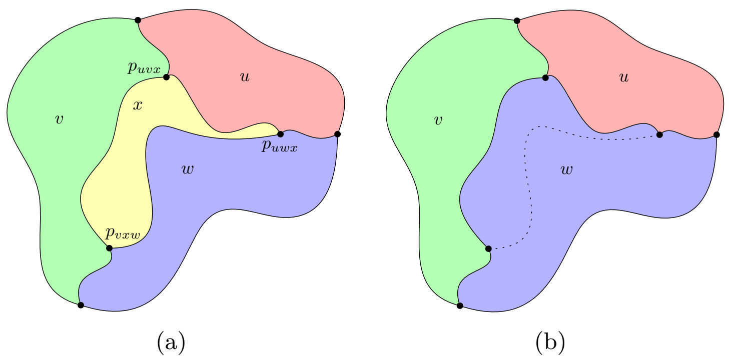


Figure 4.8: Removing an internal face x incident to the faces u, v, and w.

To remove the internal face x, we simply remove its boundary with one of its neighboring faces. In doing so, the two faces merge. In practice, this works well because large clusters do not disappear at a moment's notice, and small clusters generally occupy a minimal area such that the operation is barely noticeable.

With this trivial removal of vertices from the filtered cluster graph, the question arises why we only allow this for vertices of degree 3. Let us assume we want to remove a face that is adjacent to $n \geq 4$ other faces. By removing one of the boundaries with its neighboring faces, we would create n new adjacencies that were not there before, as illustrated in Figure 4.9. The respective edges would also need to be inserted into the cluster graph. Most importantly, though, which new adjacencies would be created depends on which one of the n boundaries is dropped. Considering future dynamic operations must remain applicable to the cluster graph and the map, we cannot allow such a non-deterministic operation. Instead, the excess edges need to be flipped away before removing the vertex is allowed, so that the operations remain deterministic.

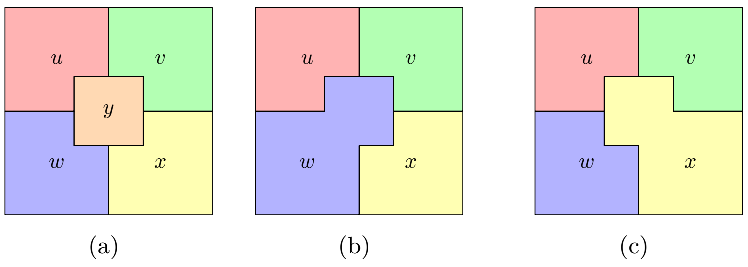


Figure 4.9: Non-deterministic removal of an internal face with four neighbors. Removing the w-y-boundary creates a w-v-adjacency (b) while removing the x-y-boundary creates an x-u-adjacency (c).

Removing External Vertices

When removing vertices on the outer face of the filtered cluster graph along with its incident edges, we must ensure that the graph remains biconnected afterward. Similar to inserting vertices on the outer face, we restrict ourselves to removing vertices on the outer face that have degree 2. If we need to remove a vertex with a higher degree, the additional edges need to be removed first, as discussed in Section 4.3. As a consequence, the operation is only permitted iff the graph has four or more vertices before the vertex' removal. Figure 4.10 shows an example of a valid removal of a vertex on the outer face.

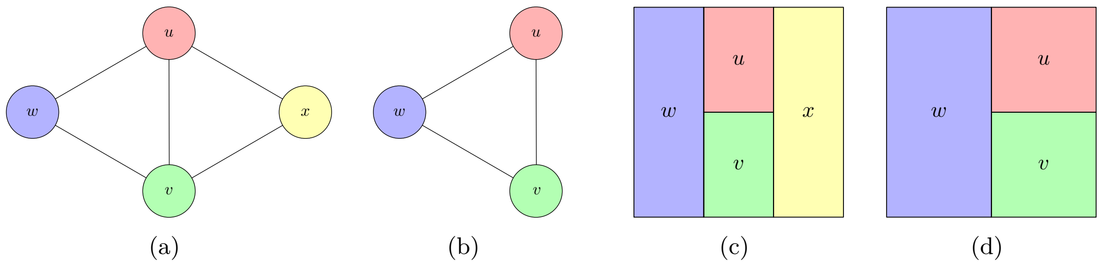


Figure 4.10: A cluster graph and a polygonal dual thereof, before (a, c) and after (b, d) removing the vertex x on the outer face.

This operation, too, is just a special case of the vertex removal discussed above. Referring to the construction of the augmented dual again, with the helper vertex v^+ and edges $\{v^+, \cdot\}$, removable vertices x on the outer face of the cluster graph lie in a triangle formed by its two neighbors u and v and the helper vertex v^+.

The construction outlined above translates 1-to-1 to removing vertices from the outer face, except one of the three neighboring faces is the implicit outer face. In our implementation, we always remove the boundary of x with the outer face and thereby transfer the area to the outer face.

4.3 Flipping Edges

Let us now discuss the edge flip mentioned in the previous sections. An internal edge $\{u, v\}$ is incident to two different internal faces f, g. Let x and y denote the third vertex bounding f and g, respectively. It is $x \neq y$ because the cluster graph is simple. Flipping the edge $\{u, v\}$ would replace it with the edge $\{x, y\}$. Consequently, this operation is only permitted iff x and y are not already adjacent — otherwise, we would introduce a duplicate adjacency. Figure 4.11 shows an example of a valid edge flip operation.

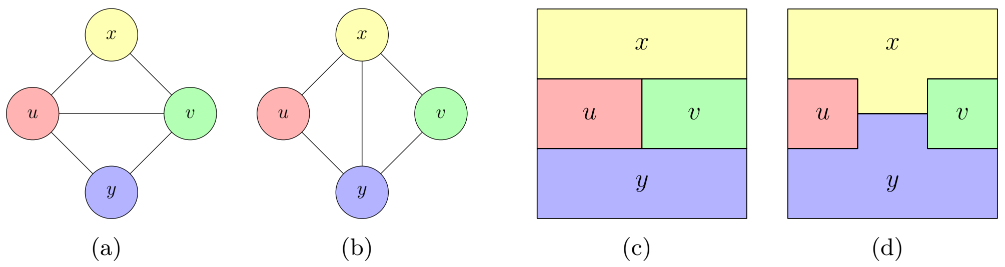


Figure 4.11: A cluster graph and a polygonal dual thereof, before (a, c) and after (b, d) flipping the internal edge $\{u, v\}$.

An edge flip in a cluster graph translates to region adjacencies being flipped in its dual. Given a polygonal dual of some cluster graph, we apply an edge flip in two phases. First, we contract the region boundary we want to remove into a single point, creating a degenerate contact representation in which four regions meet in a point. In the second phase, we create a region boundary in the opposite direction, getting rid of the degeneracy at the point into which the original boundary has been contracted.

Let u and v be two adjacent faces in the polygonal dual whose boundary we want to contract. Also, let path P_{uv} be the maximal common boundary between u and v, oriented such that u lies on the left of it, and v lies on the right of it. At both endpoints of the path, u and v meet with a third face. We denote the third face incident to the path's first vertex by x and the third face incident to the path's last vertex by y, as illustrated in Figure 4.11. To contract the u-v-boundary into a single point, we repeatedly contract a peripheral edge on the boundary until the last edge has been contracted. We do so on alternating ends, i.e., we start by contracting the first edge, then the last, then the first again, etc.

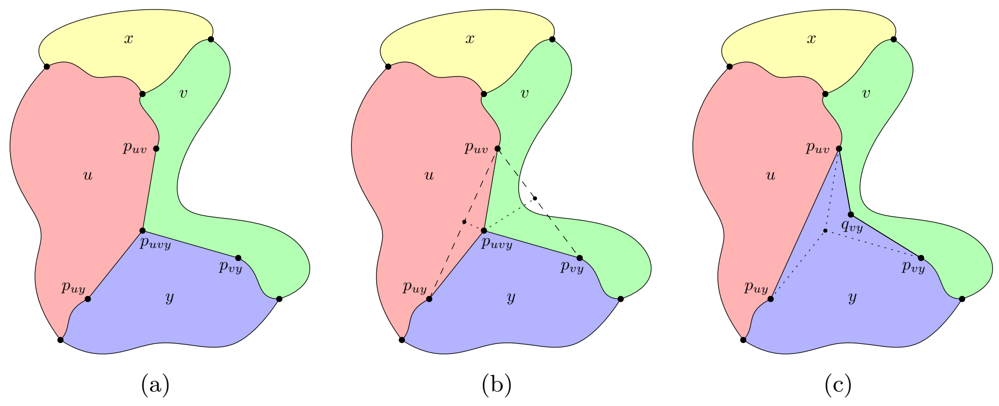


Figure 4.12: A contact representation before (a) and after (c) contracting the peripheral edge $\{p_{uv}, p_{uvy}\}$ on the u-v-boundary away from y. (b) shows the construction of potential subdivision vertices.

Figure 4.12 illustrates how we contract the first edge of the oriented upwards without introducing edge crossings. We describe only this construction in writing; the contraction of the last edge works virtually the same way and is illustrated in Figure 4.13. Let p_{uvy} denote the vertex where the faces u, v, and y meet and p_{uy} and p_{vy} the subdivision vertices on the u-y- and v-y-boundaries that are incident to p_{uvy}, respectively. If the u-y- or v-y-boundary consists of only one edge, we subdivide it at its midpoint first. Let p_{uv} be the subdivision vertex on the u-v-boundary that is incident to p_{uvy} or the last vertex of the oriented boundary if no such subdivision vertex exists. To reduce the length of the u-v-boundary by one, we would want to remove p_{uvy} and its incident edges and add edges from p_{uv} to both p_{uy} and p_{vy}. These edges may introduce crossings, though, as illustrated by the dashed lines in Figure 4.13(b) and Figure 4.12(b). However, with just one bend on each of the edges, we can guarantee that no edge crossings are created:

- If adding the edge between p_{uv} and p_{ay} ($a \in \{u, v\}$) does not introduce a crossing, we simply add the edge. (for $a = u$ in Figure 4.12(b))
- Otherwise, if the internal angle of face a at p_{uvy} is 180° or more, we place the bend at p_{uvy}, i.e., we insert the edge $\{p_{uv}, p_{ay}\}$ and subdivide it with a new vertex q_{ay} at the position of p_{uvy}. Note that at most one of the faces can have an internal angle at p_{uvy} that is 180° or more. (for $a = v$ in Figure 4.13(b))
- Otherwise, we search for a bend location in the form of a subdivision vertex q_{ay} somewhere on the outward-pointing bisector of the angle $\angle_{p_{ay}p_{uvy}p_{uv}}$ (dotted lines in Figure 4.12(b) and Figure 4.13(b)). We start looking at the point where the bisector intersects the segment from p_{uv} to p_{ay} and repeatedly divide the remaining distance to p_{uvy} in half until we find a bend location for which the bent edge from p_{uv} to p_{ay} would not introduce edge crossings. As the candidate location moves infinitesimally close to p_{uvy}, we are guaranteed to find one that does not introduce crossings. (for $a = v$ in Figure 4.12(c) and $a = u$ in Figure 4.13(c))

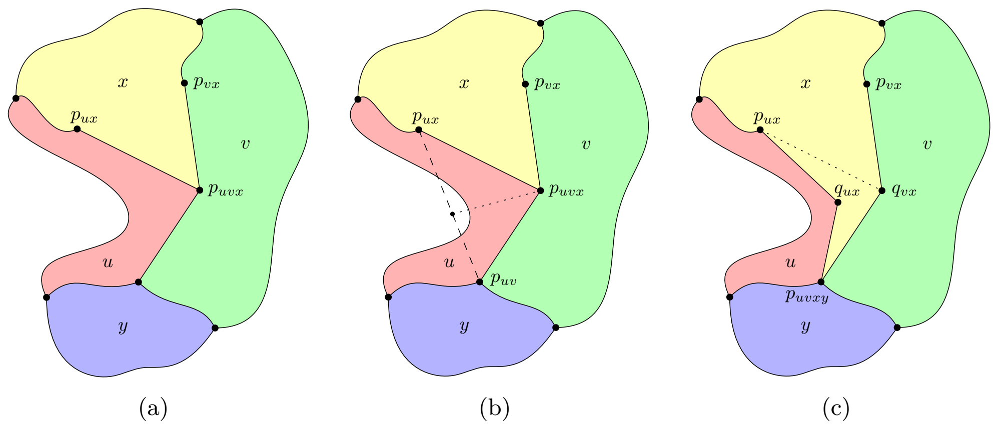


Figure 4.13: A contact representation before (a) and after (c) contracting the last remaining edge $\{p_{uvx}, p_{uvy}\}$ on the u-v-boundary away from y. (b) shows the construction of potential subdivision vertices.

Once the u-v-boundary has been contracted into a single vertex p_{uvxy} where all four faces u, v, x, and y now meet, as shown in Figure 4.13(c), we need to resolve the degeneracy and create a boundary in the opposite direction, i.e., an x-y-boundary. Let p_{ux}, p_{uy}, p_{vx}, and p_{vy} denote the subdivision vertices on the respective boundaries that are incident to p_{uvxy}. Again, if a boundary consists of only one edge, we subdivide it at its midpoint first such that all p_{ab} exist. We are now looking to stretch the vertex p_{uvxy} back into a non-degenerate x-y-boundary, as illustrated in Figure 4.14. Analogous to above, we search for a position in face u (v) where we can place a vertex q_u (q_v) and add edges to p_{uvxy}, p_{ux}, and p_{uy} (p_{uvxy}, p_{vx}, and p_{vy}) without introducing crossings. We start at the intersection of the segment between p_{ux} and p_{uy} (between p_{vx} and p_{vy}) and the bisector of $\angle_{p_{ux}p_{uvxy}p_{uy}}$ ($\angle_{p_{vx}p_{uvxy}p_{vy}}$) and repeatedly divide the distance to p_{uvxy} in half until we find a valid position. Once we have found such a position, we insert the vertex q_u (q_v) along with the aforementioned edges and remove the edges $\{p_{uvxy}, p_{ux}\}$ and $\{p_{uvxy}, p_{uy}\}$ ($\{p_{uvxy}, p_{vx}\}$ and $\{p_{uvxy}, p_{vy}\}$). In case $\measuredangle_{p_{ux}p_{uvxy}p_{uy}} \geq 180°$ ($\measuredangle_{p_{vx}p_{uvxy}p_{vy}} \geq 180°$), we will not find such a position and shall not make an adjustment on that side. However, again, this event cannot occur for both u and v at the same time. As a result, we always create an x-y-boundary that consists of at least one edge.

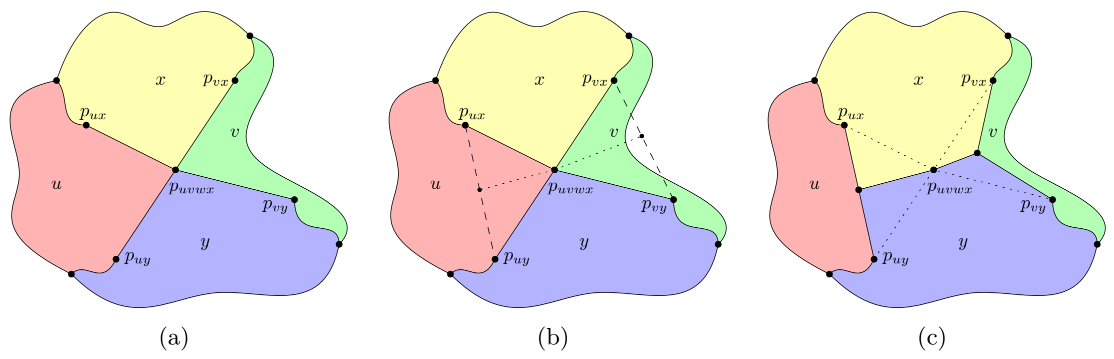


Figure 4.14: A contact representation before (a) and after (c) creating a non-degenerate yellow-blue adjacency. (b) shows the construction of potential subdivision vertices.

Inserting and Removing Edges

As clusters in our data set grow increasingly similar, we may want to indicate this similarity with new edges between existing clusters in the cluster graph. Similarly, clusters can grow apart, and we may want to remove edges between clusters in the cluster graph. However, because the filtered cluster graph is internally triangulated, we cannot insert any more edges on the inside of the graph. Removing an internal edge is not permitted either, as that would create a hole in the graph. Consequently, we can insert edges only in the outer face and remove edges only on the outer face.

On top of that, inserting an edge in the outer face is only possible if it preserves the cluster graph's internal triangulatedness. Therefore, inserting an edge $\{u, w\}$ is only permitted iff u and w lie on the outer face and have a neighbor v in common that also lies on the outer face. The inserted edge is then embedded such that it forms a new triangular internal face with v and turns v into an internal vertex. If exactly four vertices bound the outer face before inserting the edge $\{u, w\}$, there exist two candidates for v. In this case, it must be made explicit which one is supposed to become internal and which one is supposed to remain on the outer face. Figure 4.15 shows an example of a valid edge insertion.

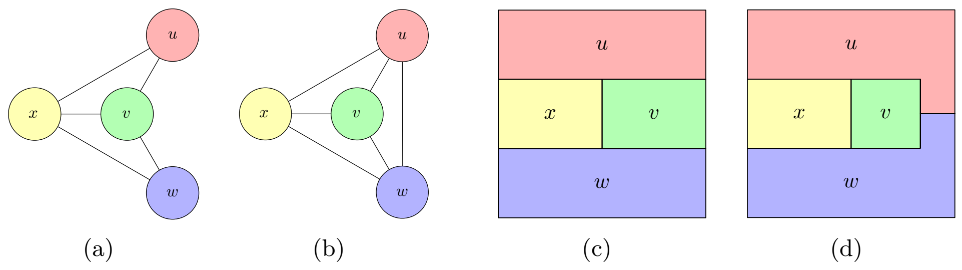


Figure 4.15: A cluster graph and a polygonal dual thereof, before (a, c) and after (b, d) inserting the edge $\{u, w\}$ to form an internal triangular face with v.

Removing an edge $\{u, w\}$ on the outer face of the cluster graph is only permitted if the graph remains biconnected. This property is preserved iff both u and w have degree $d(\cdot) \geq 3$ and the third vertex v in the internal face bounded by $\{u, w\}$ does not already lie on the outer face. If that vertex laid on the outer face already, we would end up creating a duplicate adjacency/boundary between v and the outer face, which we specifically excluded in Chapter 2. Figure 4.16 shows an example of a valid edge removal.

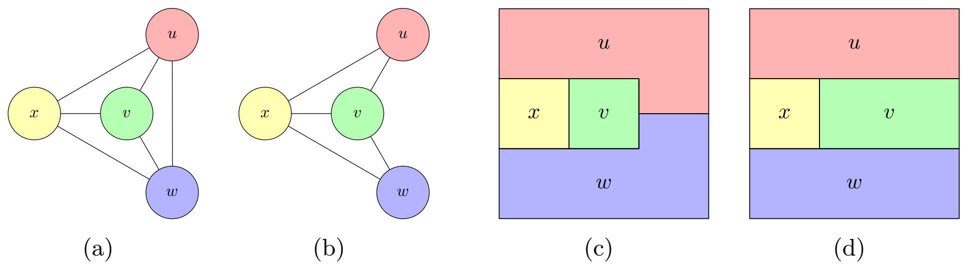


Figure 4.16: A cluster graph and a polygonal dual thereof, before (a, c) and after (b, d) removing the edge $\{u, w\}$.

Both of these operations are again a special case of the generic edge flip discussed above with a quirk: Instead of four internal faces, they involve three internal faces and the implicit outer face.

In the example from Figure 4.15, we make v into an internal vertex by inserting the edge $\{u, w\}$, removing v's boundary with the outer face in the dual and creating a u-w-boundary in turn. Again, recall that in the construction of the augmented dual from Definition 3.3, we insert a helper vertex v^+ in the outer face and add edges to all vertices on the original outer face. This helper vertex and its adjacencies correspond to the outer face and its boundaries in the dual. When inserting an edge in the outer face, we are essentially flipping the helper edge $\{v, v^+\}$ to become the inserted edge $\{u, w\}$. Therefore, we apply the same procedure as discussed for the edge flip above, contracting the boundary between v and the outer face into a single point and then creating a non-degenerate u-w-boundary.

Similarly, when removing an edge $\{u, w\}$ on the outer face as in Figure 4.16, a previously-internal vertex v moves onto the outer face. We can think of it as flipping the edge to be removed to become the helper edge $\{v, v^+\}$. In the polygonal dual, this gets rid of the u-w-boundary and creates a boundary between v and the implicit outer face in turn. Again, we apply the same procedure as discussed for the edge flip, first contracting the u-w-boundary into a single point and then creating a non-degenerate boundary between v and the outer face.

5. Evaluation

In this chapter, we experimentally evaluate our implementation of the framework discussed in Chapters 3 to 4. The following research questions drive our evaluation:

1. Which quantitative measures best capture the quality of the maps generated by our algorithm in terms of
    a) accuracy?
    b) our understanding of locally fat regions?
2. What is the quality of the maps generated by our algorithm according to these quality metrics?
    a) How does this quality change based on the size and other properties of the input graph?
    b) How does this quality change over time as dynamic updates are incorporated?

First, we discuss a variety of possible quality metrics from literature and how they can be applied to our framework in Section 5.1. We pick two metrics that best capture the quality of the generated maps according to research question 1. In Section 5.2, we present the algorithm we use to generate randomized test instances on which we perform our experimental evaluation. In Sections 5.3 to 5.4, we conduct the actual experimental evaluation on a large number of test instances and see how the maps generated by our algorithm perform, thereby answering research question 2. Eventually, in Section 5.5, we showcase a range of maps produced by our framework.

5.1 Quality Metrics

We start by looking at common quality metrics of cartograms — which are closely related to the problem we are trying to solve in this thesis, as previously discussed in Section 1.1 — and discussing how they translate to the visualizations our framework creates. We then discuss different ideas to formalize and quantify the previously mentioned notion of the region's local fatness.

Quality Metrics of Cartograms

Three well-established quantifiable measures are commonly used to assess a cartogram's quality [AKV15] [NAK18]:

- **Statistical accuracy:** The statistical accuracy of a cartogram describes how closely the areas of the modified geographic regions match the variable of interest. Recall that the normalized cartographic error of a region v is defined as $\frac{|A'(v)-w(v)|}{max(A'(v),w(v))}$, where $A'(v)$ is its (normalized) actual area, and $w(v)$ is its desired area. The maximum and average normalized cartographic error over all cartogram regions is commonly used to quantify its statistical accuracy. We borrow this quality metric from cartograms as we have already used it previously to define ε-proportional maps.
- **Topological accuracy:** The topological accuracy describes how well the original adjacencies between the geographic regions are preserved in the cartogram. Considering our framework computes a polygonal contact representation of the filtered cluster graph, those and only those regions whose corresponding vertices are adjacent in the cluster graph are adjacent in the contact representation. In terms of topological accuracy, we therefore produce perfect drawings.
- **Geographic accuracy:** The geographic accuracy of a cartogram describes the degree to which the shapes and positions of the distorted regions resemble their original counterpart on the real geographic map. In our case, however, there is no real geographic map that our visualization is based upon: The maps we generate are entirely artificial, and there is no geographic reference map.

  However, the motivation behind geographic accuracy as a quality metric still makes sense in our setting. Geographic accuracy captures the preservation of the viewer's mental map between the real geographic map and the distorted map in the cartogram. This idea applies to our framework as well, albeit only once we start incorporating dynamic updates into the artificial map. Our framework naturally preserves the viewer's mental map for dynamic inputs by only allowing small, incremental changes to be incorporated, and by redrawing the artificial map using a force-directed algorithm. This approach makes it easy for the viewer to track how the map changes between versions of the underlying cluster graph at different points in time.

Polygon Complexity

Brinkhoff et al. [BKSB95] propose a method to quantify the complexity of polygons. They measure the complexity of polygons as a combination of three properties: the frequency of the vibration of its boundary, the amplitude of said vibration, and the deviation from its convex hull. Given a polygon P, they count the number of corners where the polygon's internal angle is more than 180°, called *notches*, and define the fraction of those corners as

$$\text{notches}_{[0,1]}(P) := \begin{cases} \frac{\text{notches}(P)}{n-3} & \text{if } n > 3 \\ 0 & \text{otherwise} \end{cases} \in [0,1],$$

where n is the total number of corners in P and notches(P) is the number of notches in P. They use this fraction to define the frequency of the vibration as

$$\text{freq}(P) := 1 + 16 \cdot (\text{notches}_{[0,1]}(P) - 0.5)^4 - 8 \cdot (\text{notches}_{[0,1]}(P) - 0.5)^2 \in [0,1]$$

and its amplitude as

$$\text{ampl}(P) := \frac{\text{circumference}(P) - \text{circumference}(\text{hull}(P))}{\text{circumference}(P)} \in [0,1],$$

where circumference(P) is the length of P's boundary, and hull(P) is the convex hull of P's corners in the form of another polygon. These equations were chosen such that the frequency freq(P) of the vibration of a polygon reaches its maximum when half of the polygon's corners are notches, and the amplitude ampl(P) of the vibration is close to 1 when the area of the polygon is very small in relation to the area of its convex hull. Intuitively, for convex polygons P, both the frequency freq(P) and the amplitude ampl(P) of the vibration of its boundary is zero.

To measure the convexity of a polygon, Brinkhoff et al. use the fraction of the area of the polygon's convex hull that the polygon itself covers:

$$\mathrm{conv}(P) \coloneqq \frac{\mathrm{area}(\mathrm{hull}(P)) - \mathrm{area}(P)}{\mathrm{area}(\mathrm{hull}(P))} \in [0,1]$$

According to their observations, a relative increase of the boundary in combination with a high-frequency vibration of the boundary has the most significant impact on the intuitive perception of a polygon's complexity [BKSB95]. They therefore combine these three properties as

$$\mathrm{compl}(P) \coloneqq 0.8 \cdot \mathrm{ampl}(P) \cdot \mathrm{freq}(P) + 0.2 \cdot \mathrm{conv}(P) \in [0,1] \tag{5.1}$$

to measure the overall complexity of a polygon.

We adopt this definition of polygon complexity as a quality metric for the maps generated by our framework, but only with a slight change because, as discussed above, compl$(\cdot)$ assigns a perfect score of 0 to all convex polygons. This means that, for example, there is no distinction between a square region and a long, drawn-out rectangular region, even though the square region matches our understanding of local fatness much better. Instead of calculating the fraction of the area that a polygon P does not cover within its convex hull, we therefore compute the fraction of area it does not cover in its smallest enclosing circle. To keep conv$(\cdot)$ in the unit interval, we compare P's area to the maximal area of a regular n-gon in said smallest enclosing circle:

$$\mathrm{compl}'(P) \coloneqq 1 - \frac{\mathrm{area}(P)}{\mathrm{area}(\mathrm{smallestEnclosingCircle}(P)) \cdot \sin\left(\frac{360^\circ}{n}\right) \cdot \frac{n}{2\pi}} \in [0,1]$$

In our tests, this measure has shown to align with our intuitive understanding of local fatness nicely.

Entropy

Chen and Sundaram [CS05] studied the complexity of 2-dimensional shapes in the field of computer vision. The shapes they dealt with are given in the form of a point cloud. For each point, they compute the Euclidean distance to the point cloud's centroid. They also implement a heuristic to predict the contour of the shape at each point and use it to compute a local angle for each of the points. The distances and local angles are then plotted into a histogram with flexible bin size and are used to compute the entropy of the distribution of distances and local angles.

In our use case, we do not even have to implement the heuristic to guess the shape's contour. We know that the points form a closed curve that is, in fact, a polygon. However, by connecting the points in this predetermined manner, we possibly get drastically different edge lengths. The points on these edges that are not corners of the polygon have no impact on the computed entropies. This problem becomes obvious when we consider two straight-line segments of the same length, one that is a single side of our polygon, and one which is the union of multiple sides of our polygon with corners (with internal angle 180°) in between: The one with additional corners has a much greater impact on the computed entropies, even though both look the same to an observer.

We can try to remedy by subdividing the polygon's sides first, creating roughly uniform side lengths. However, in doing so, we create lots of corners where the polygon has an internal angle of 180°, quickly dominating and distorting the entropy of internal angles.

In our tests, neither of two entropy-based measures, with or without our adjustments, captured our intuitive understanding of local fatness satisfactorily. We therefore conduct our experimental evaluation in Section 5.4 using just cartographic error and our modified version of polygon complexity from Brinkhoff et al. [BKSB95] discussed above. For both measures, we will look at both the maximum and average values over all regions of the map.

5.2 Test Case Generation

We implement a randomized test case generation in two phases. First, we generate a filtered cluster graph $G^{\mathcal{C}}$. We accompany this cluster graph with a planar straight-line drawing $\Gamma^{\mathcal{C}}$ thereof so that such a drawing does not need to be calculated in retrospect, as previously discussed in Section 3.1. Second, we generate a sequence of dynamic operations to be applied to said cluster graph.

Filtered Cluster Graph Generation

Our test case generation can be configured using the following five parameters, the usage of which becomes evident in the following:

- **Number of clusters:** The number of clusters $n \in \mathbb{N}, n \geq 3$ determines how many vertices the generated cluster graph $G^{\mathcal{C}}$ has.
- **Bounding box:** The axis-aligned bounding box $\mathcal{A} = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ determines the area in which the cluster graph's vertices can be placed.
- **Weight distribution:** The cluster weight distribution is given as a probability mass function $\mathrm{pmf}\colon \mathbb{N}_+ \to \mathbb{R}_+$ and determines how the cluster weights are distributed.
- **Nesting ratio:** The nesting ratio $\alpha \in [0, 1]$ determines what fraction of the n vertices are nested into other triangular faces.
- **Nesting bias:** The nesting bias $\beta \in [0, 1)$ determines to what degree nesting vertices into already-nested triangular faces is preferred over nesting vertices into top-level triangles.

The idea behind generating filtered cluster graphs $G^{\mathcal{C}}$ is as follows: We start by randomly placing a fixed number of vertices within the bounding box $\mathcal{A}$ and computing a triangulation of these vertices. We then place the remaining vertices into existing triangles based on the nesting ratio α and bias β, inserting additional edges to those triangles' endpoints along the way. The algorithm is illustrated in pseudocode in Algorithm 5.1 on the next page.

The generated graph is obviously plane and internally triangulated. Considering a point set's triangulation covers the same area as the set's convex hull, no vertex of the generated graph can appear on its outer face more than once. Therefore, the graph is also biconnected.

Due to the independent sampling of vertex positions in Algorithm 5.1, vertices often clump together, especially for larger test instances. This is not an ideal starting position for the transformation as previously discussed in Section 3.1 because the polygonal dual contains at least twice the number of vertices, therefore clumping even more and severely restricting the involved vertices' movement. To address this issue, we apply a force-directed "shake" to the generated straight-line drawing $\Gamma^{\mathcal{C}}$ of the cluster graph while preserving its edge crossing and combinatorial properties.

We do this by defining attractive forces between pairs of adjacent vertices and strong repulsive forces between non-adjacent vertices and between edges and non-incident vertices based on their distances. Again, we apply the rules of ImPrEd [SAAB11] to prevent the introduction of edge crossings. The repulsive forces are the same as the ones defined in Section 3.2, although we use a higher scaling constant of 1000 here to really prevent the vertices from clumping together. For the attractive force between adjacent vertices, we apply the force $F_{\text{att}}(u, v) := \log(d(u, v)/100)$ to both endpoints of an edge, directed towards each other. Here, the value of 100 represents our ideal edge length. This new force is based on logarithmic springs, as suggested by Eades [Ead84].

Algorithm 5.1: Randomized Filtered Cluster Graph Generation

```
Input: bounding box $\mathcal{A}$, weight probability mass function $\mathrm{pmf}(\cdot)$, count $n$, nesting
       ratio $\alpha$, nesting bias $\beta$
Output: planar straight-line drawing $\Gamma^{\mathcal{C}}$ of a filtered cluster graph $G^{\mathcal{C}}$ on $n$
        vertices
1  create empty straight-line drawing $\Gamma^{\mathcal{C}}$
2  $k \leftarrow \min(\lfloor \alpha \cdot n \rfloor, n-3)$ // number of nested vertices
   // sample $n-k$ pairwise distinct points $p_i \in \mathcal{A}$
3  foreach index $i \in [0, n-k) \cap \mathbb{N}$ do
4      sample random point $p_i$ within $\mathcal{A}$ (uniformly)
5      sample random weight $w_i$ (according to $\mathrm{pmf}(\cdot)$)
6      add vertex $i$ with weight $w_i$ at position $p_i$ to $\Gamma^{\mathcal{C}}$
   // create edges according to Delaunay triangulation of points
7  foreach triangle $\triangle_j = (u, v, w) \in$ Delaunay triangulation of points $(p_i)_i$ do
8      foreach $(a, b) \in \{(u, v), (v, w), (w, u)\}$ do
9          insert edge between $a$ and $b$ in $\Gamma^{\mathcal{C}}$ unless edge already exists
10     register triangular face $\triangle_j$ with $\mathrm{depth}(\triangle_i) = 1$
   // nest remaining $k$ vertices into existing triangles
11 foreach index $i \in [n-k, k) \cap \mathbb{N}$ do
12     sample random triangle $\triangle_i$ (weighted by $(1-\beta)^{-\mathrm{depth}(\cdot)}$)
13     sample random point $p_i$ within $\triangle_i$ (uniformly)
14     sample random weight $w_i$ (according to $\mathrm{pmf}(\cdot)$)
15     add vertex $i$ with weight $w_i$ at position $p_i$ to $\Gamma^{\mathcal{C}}$
16     add edges between $i$ and all of $\triangle_i$'s endpoints to $\Gamma^{\mathcal{C}}$
17     unregister triangular face $\triangle_i$
18     register the three new triangular faces with depth $1 + \mathrm{depth}(\triangle_i)$
19 return $\Gamma^{\mathcal{C}}$
```

Dynamic Operation Generation

We generate sequences of dynamic operations σ_t to be incorporated into the cluster graph $G_t^{\mathcal{C}}$ and the resulting map $\Gamma_t^{\propto}$ at different points in time t. In such a sequence of operations, we combine simple weight change operations with topology-altering operations, namely inserting or removing vertices or edges, or flipping internal edges in the cluster graph $G_t^{\mathcal{C}}$.

For each point in time t at which we want to incorporate dynamic operations, we first sample new target weights w_i' for all clusters i according to $\mathrm{pmf}(\cdot)$. To prevent too drastic weight changes that would likely not appear in natural data sets, we set the clusters' new weights to a weighted average of their current weight and the newly sampled target weight as $w_{i,t+1} \coloneqq \frac{3}{4} w_{i,t} + \frac{1}{4} w_i'$. Finally, we compute the set of valid operations affecting topology and pick one at random. To prevent instances from getting bigger and bigger due to there being way more valid vertex insertion operations, we first pick a non-empty class of operations uniformly at random and then select a random operation inside that class uniformly at random. We apply this topology-altering operation alongside the weight changes in σ_t. Each σ_t therefore consists of $|V(G_t^{\mathcal{C}})| + 1$ dynamic operations.

5.3 Implementation and Experiment Setup

Besides this written report, we have implemented the framework illustrated in Chapters 3 to 4, along with the test case generation discussed in Section 5.2, as a Swift package. We also provide a graphical user interface to explore the individual steps of the pipeline interactively.

To perform the experimental evaluation, we include a command-line tool that efficiently generates test instances, runs them through the pipeline, and assesses their quality. Each instance is generated randomly based on the parameters n, α, and β discussed previously; the bounding box $\mathcal{A}$ and the weight distribution pmf are kept constant. We vary n, α, and β to get insights into their effect on the generated maps' qualities. Whenever we run the force-directed optimization algorithm, we perform $10c$ iterations. Here, c is the current number of clusters in the cluster graph or the number of regions in the polygonal dual.

The implementation is open source and can be found on GitHub:

`https://github.com/jenox/master-thesis/`

The results presented here are based on 100 randomly generated identifiers that are included in the repository. We use these identifiers to seed the pseudorandom number generator used by our framework, thereby making the results presented here reproducible on any machine running macOS 10.15 or later.

5.4 Experimental Evaluation

We create various plots showing the distribution of the maps' qualities to gain insights into the effect of the number of clusters n, the nesting ratio α, nesting bias β, and the number of dynamic operations t on the different quality metrics. In these plots, we vary the value for one of these parameters, while keeping the others constant.

Figure 5.1 shows how the number of dynamic operations t applied to the instance affects the cartographic error and polygon complexity.

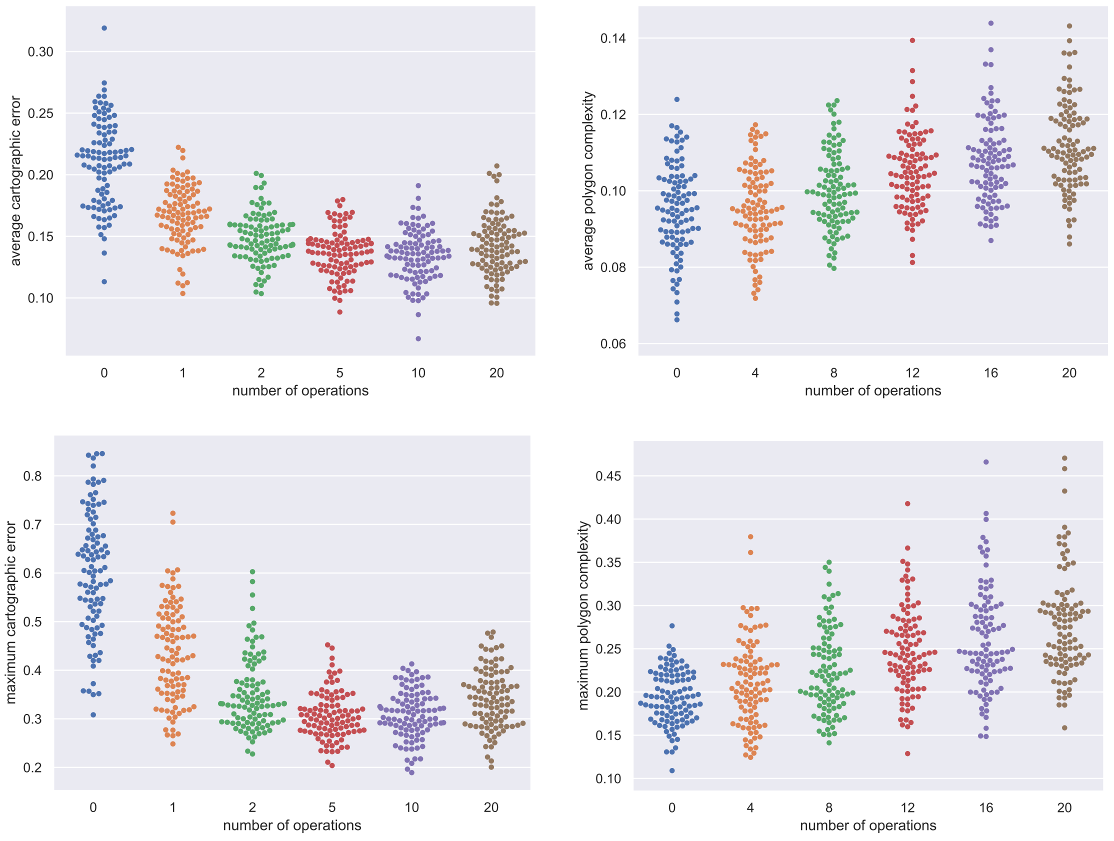


Figure 5.1: Quality metrics of maps $\Gamma_t^{\propto}$ for 100 randomized instances after applying a different number of operations t, with $n = 20$, $\alpha = 0$, and $\beta = 0$.

We can see that as we apply more operations, the cartographic error decreases, and very quickly so: after $t = 5$ operations, the cartographic errors become almost indistinguishable. We believe that this is because, for larger values of t, the force-directed optimization algorithm has more time overall to optimize the statistical accuracy of the maps, amongst other features.

The polygon complexity, on the other hand, increases over time. When processing topology-altering operations, additional subdivision vertices may be inserted to prevent the introduction of edge crossings, potentially creating shapes that are not locally fat in the process. The optimization algorithm obviously tries to improve the local fatness of the involved regions afterward. However, it seems unable to do so until around $t = 16$, where it seems to become able to counteract these effects and to prevent further degradation. In Section 6.1, we discuss an approach to verify whether this change indeed happens over time, or lies in the nature of the arrangement of the regions in the different test instances.

The maximum cartographic error and polygon complexity over all regions of the map paint pictures with similar trends but greater mean and variance as outliers have a more significant impact on the map's overall quality.

The effect of the initial number of clusters n on the maps' quality is shown in Figure 5.2.

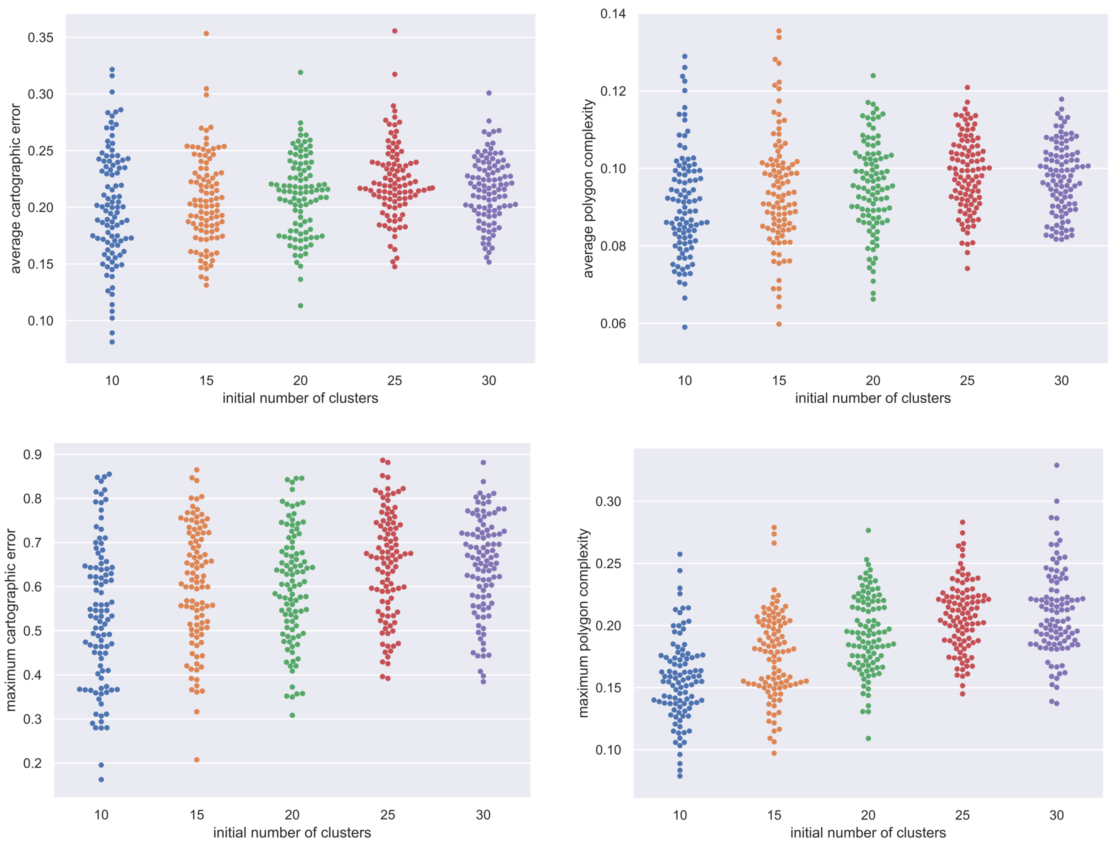


Figure 5.2: Quality metrics of maps $\Gamma_t^{\propto}$ for 100 randomized instances with different numbers of clusters n, with $\alpha = 0$, $\beta = 0$, and $t = 0$.

The average cartographic error across all instances appears mostly unaffected by the input size. Its variance decreases, though, as we increase the number of clusters. This observation makes sense because as there are more clusters, the impact an outlier has on the average cartographic error decreases.

Regarding the polygon complexity, a slight increase in complexity for larger input sizes can be seen in the plot. We believe that this is because, for larger instances, there is simply more room for challenging constructs that require higher polygon complexity to visualize correctly.

With a larger number of clusters, there come more potential outliers that can negatively affect the instances' maximum cartographic errors and polygon complexities. Hence we see more pronounced trends for these two metrics.

Figure 5.3 shows the effects of different combinations of nesting ratio α and nesting bias β on the maps' quality.

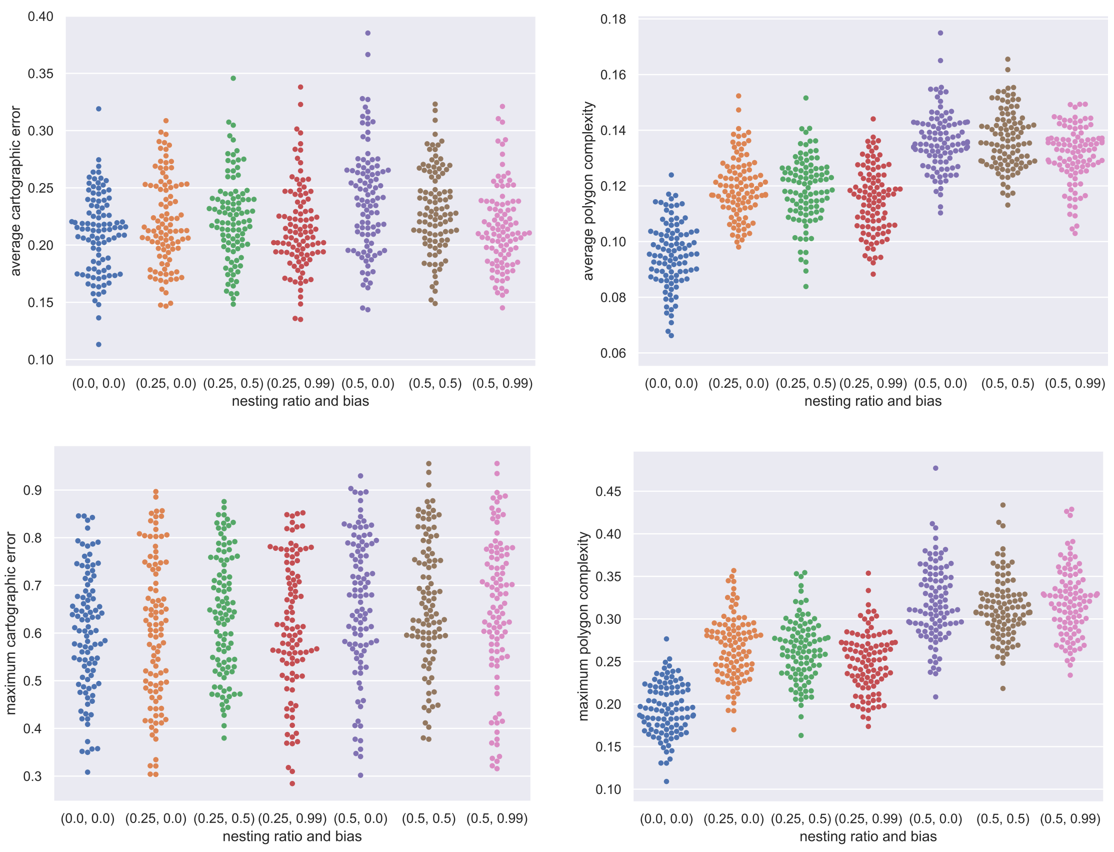


Figure 5.3: Quality metrics of maps $\Gamma_t^{\propto}$ for 100 randomized instances with different nesting ratios α and nesting biases β, with $n = 20$ and $t = 0$.

We cannot see any meaningful effect of nesting ratio and bias on the maps' cartographic error.

For the polygon complexity, however, the nesting ratio α has a pronounced impact. The figure clearly shows that instances with $\alpha = 0.25$ tend to have higher polygon complexities than instances with $\alpha = 0$, and instances with $\alpha = 0.5$ even higher ones, regardless of nesting bias β.

A possible explanation is that by nesting vertices of the cluster graph into existing triangles, we create internal vertices with low degrees. Therefore, the regions of the map corresponding to these vertices are surrounded by few neighboring regions, which are inevitably more complex as these few regions have to span the entire boundary of the original region. Additionally, there generally lie fewer vertices on the outer face of the cluster graph, whose corresponding regions must again surround the entire inside. These decreases in the number of external vertices and average degree of internal vertices as the nesting ratio α increases are illustrated in Figure 5.4.

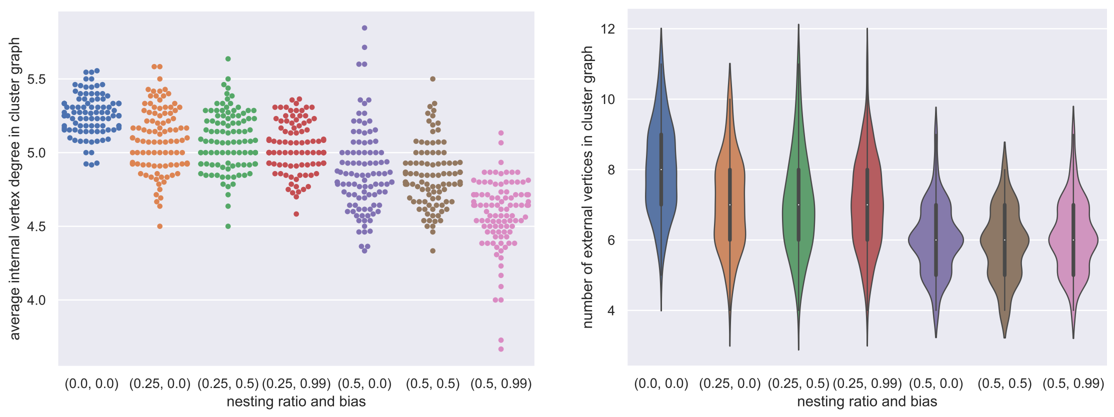


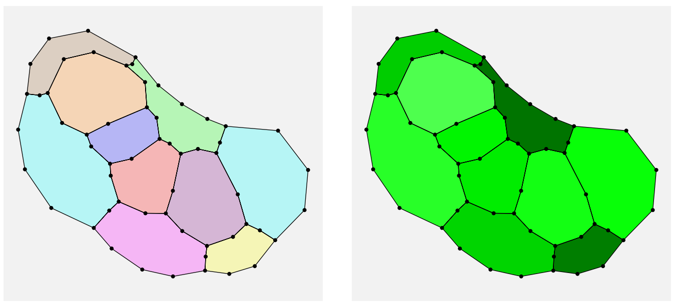
Figure 5.4: Properties of 100 randomized filtered cluster graphs $G_t^{\mathcal{C}}$ with different nesting ratios α and nesting biases β, with $n = 20$ and $t = 0$.

Figure 5.3 also indicates that for fixed a nesting ratio α, the nesting bias β slightly reduces the map's polygon complexity. However, this observation is not significant enough to jump to any conclusions.

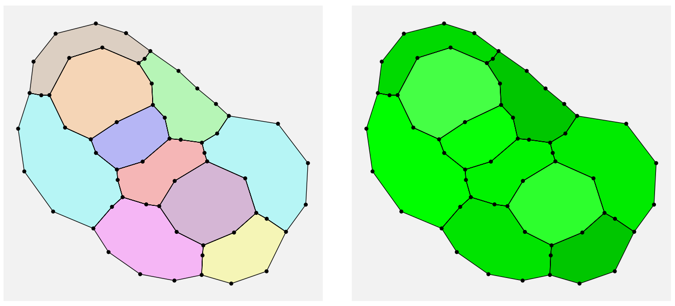
Again, the maximum polygon complexity over all regions of the map exhibits the same overall trend but with a larger spread.

5.5 Exemplary Maps

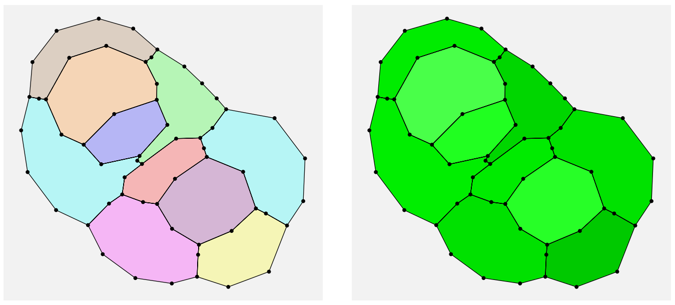
In this section, we showcase a range of maps produced by our framework. Figure 5.5 shows how such a map evolves over time as the underlying cluster graph changes. Figures 5.6 to 5.8 show representative maps for different numbers of regions $n \in \{10, 20, 30\}$.



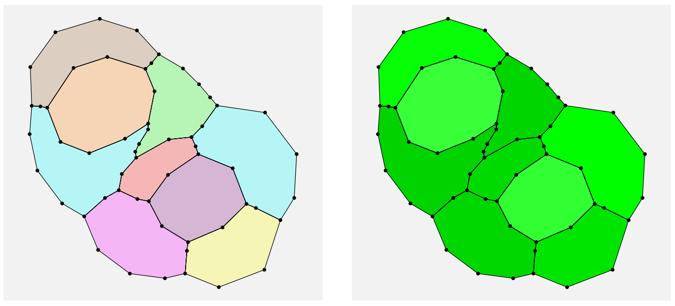
(a)



(b)



(c)



(d)

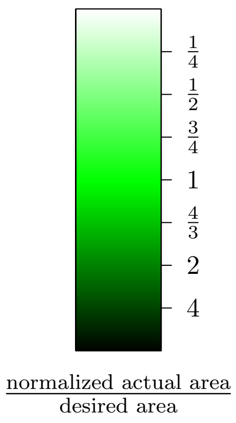


Figure 5.5: Exemplary evolution of the generated map for an input graph with $n = 10$ at different points in time $t \in \{0, 1, 2, 3\}$. The lightness of the region colors in the right column represents the pressure in the respective regions, with lighter regions having lower pressure than darker regions. From (a) to (b) the green-purple boundary is flipped, from (b) to (c) the red-blue boundary is flipped, and eventually from (c) to (d) the blue region is removed.

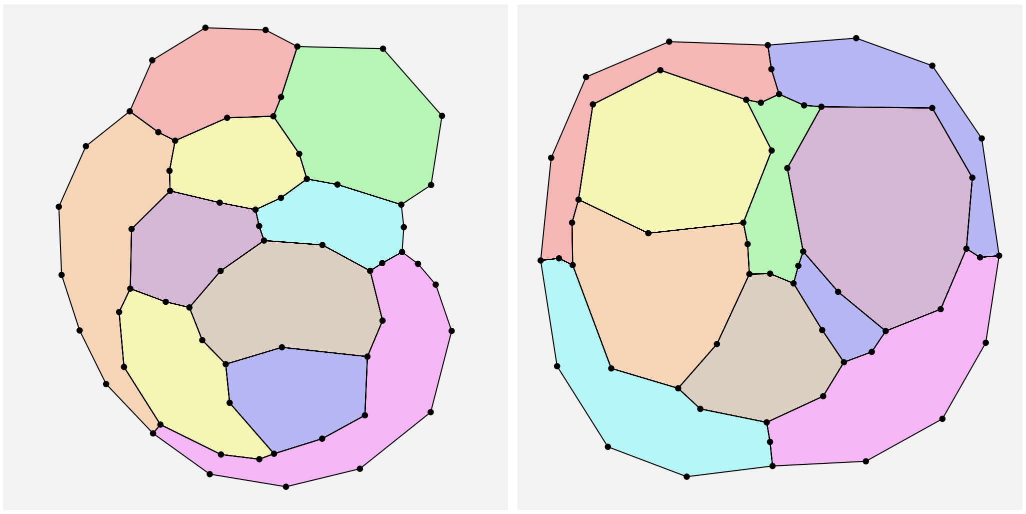


Figure 5.6: Exemplary maps produced by our framework for $n = 10$.

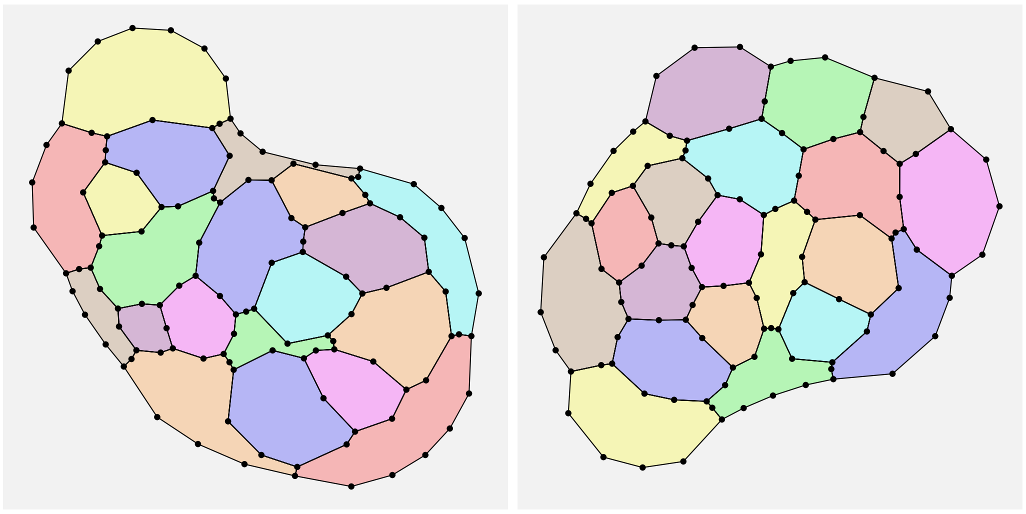


Figure 5.7: Exemplary maps produced by our framework for $n = 20$.

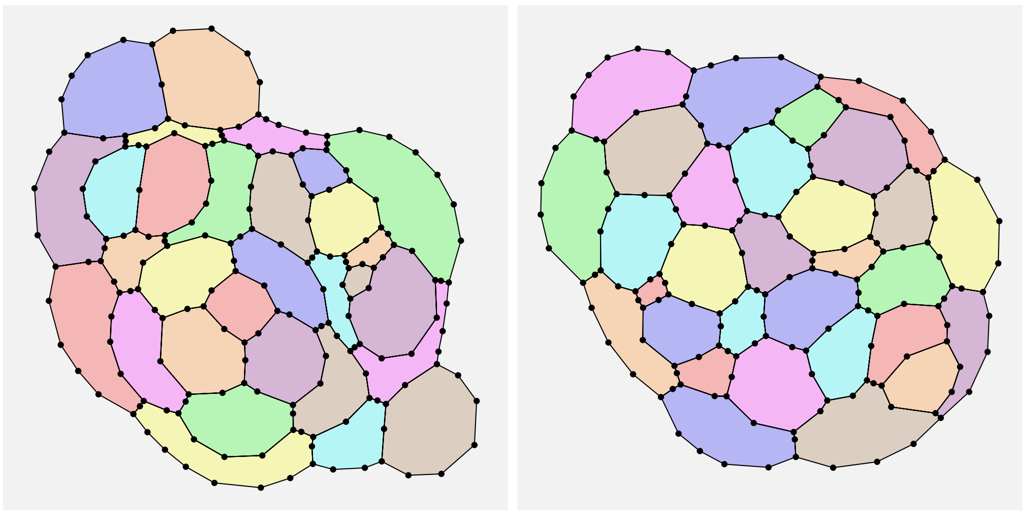


Figure 5.8: Exemplary maps produced by our framework for $n = 30$.

6. Conclusion

In this thesis, we explored how cluster information in graphs can be visualized naturally as countries on a map in a dynamic setting where the graph changes over time. We proposed a framework that produces maps whose regions' sizes are approximately proportional to the respective clusters' sizes. The framework can also react to dynamic changes of the input graph in a way that preserves the viewer's mental model of the map.

We can make these guarantees by working on an intermediate representation that encompasses all combinatorial properties of the eventual map, breaking changes down into small, incremental changes, and using a force-directed algorithm to draw the eventual map. This way, the structure of our algorithmic pipeline alone guarantees that changes to the map are minor and comprehensible, thereby preserving the viewer's mental map as the underlying data changes over time.

Besides the preservation of the mental map, related work struggles with keeping clusters as continuous regions in the generated maps [GHK09] [MKH11]. The structure of our algorithmic pipeline addresses and solves this problem, too. Considering the maps we produce are contact representations of the intermediate cluster graph, clusters cannot possibly be fragmented across different regions.

The framework presented here can be applied to a variety of real-world applications, allowing us to visualize clusters in the underlying data and how they change over time, thereby enabling viewers to detect trends in the data easily.

6.1 Future Work

Future research can improve upon several aspects of the framework discussed here.

Requirements

First and foremost, one might relax the requirements imposed on the filtered cluster graph and adjust the implementations of the different phases of the pipeline accordingly. For example, one could support connected but not necessarily biconnected cluster graphs. This relaxation would allow for additional features found on many real-world geographic maps, such as regions adjacent to only one internal face (e.g., Portugal) or regions with multiple boundaries with the implicit outer face (e.g., Spain). Similarly, one could allow cluster graphs that are not internally triangulated, allowing for maps in which lakes or rivers separate different regions. More interesting perhaps would be scenarios in which two adjacent regions do not necessarily have a single continuous boundary, but multiple ones, as is the case between Romania and Ukraine.

User Study

An empirical user study should be performed to evaluate how the quality metrics presented in Chapter 5 relate to human perception and the map metaphor. It might also be interesting to evaluate how the quality of the maps produced by our framework differs after applying long sequences of dynamic operations versus starting from scratch with the cluster graph at that point in time, i.e., not applying incremental updates to an already-existing map, but instead running the cluster graph through the regular transformation phase again. We believe that starting from scratch has the potential to get rid of artifacts that the incremental transformation and drawing phases cannot, at the cost of potentially destroying the viewer's mental map. Still, this might be worth doing every once in a while if artifacts become too pronounced.

Dynamic Operations

Regarding the dynamic operations themselves, it might be worth supporting additional, more complex operations. Such operations capture the semantics of a change much better than their decompositions into atomic operations. One might be able to implement these operations much more efficiently while also getting better results in terms of aesthetics.

Implementation

The implementations of the individual phases of the pipeline presented in Chapters 3 to 4 leave room for improvement and can be improved independently.

For example, having the transformation phase produce an initial map with zero cartographic error, which is then tweaked by the force-directed drawing phase to improve the regions' local fatness, might yield better results overall. Such an initial map exists for all planar graphs with just one bend per boundary [Kle19]. As long as one does not relax the requirements imposed on the cluster graph too much and its augmented dual remains a cubic graph, such an initial layout even exists without any bends at all [Tho92].

Similarly, one could try to improve upon the force-directed drawing phase by tweaking the forces' parameters or introducing new forces altogether. In combination with the user study mentioned above, one could improve the precise mathematical formulation of the characteristics of region shapes that we are looking for, and change the forces to work towards these characteristics accordingly.

Visualization

Like GMap [GHK09] and OpMap [vS17], one could try to draw the original, unclustered graph on top of the map. Doing so would significantly improve the visualization's expressiveness as it would show what is going on within the different clusters. However, one must pay great attention to preserve the viewer's mental map of the structure within the individual clusters as well.

It would also be interesting to see if the visual appeal of maps produced by our framework can be improved by smoothening the regions' boundaries to make the map look more organic. This effect could be achieved by computing non-intersecting splines through the polygons' corners and using those instead of the polygons' sides to bound the regions of the map.

Another idea worth exploring is using the cluster graph's edge weights to control the relative length of a region's boundaries with its neighboring regions. This way, the length of the boundary between any two regions gives the viewer an idea of how similar the respective clusters are. There exist some theoretical results on this topic [NPR12], but integrating this criterion into the force-directed formulation requires further research.

Bibliography

[ABF$^+$13a] Muhammad Jawaherul Alam, Therese Biedl, Stefan Felsner, Andreas Gerasch, Michael Kaufmann, and Stephen G. Kobourov. Linear-time algorithms for hole-free rectilinear proportional contact graph representations. *Algorithmica*, 67(1):3–22, 2013.

[ABF$^+$13b] Muhammad Jawaherul Alam, Therese Biedl, Stefan Felsner, Michael Kaufmann, Stephen G. Kobourov, and Torsten Ueckerdt. Computing cartograms with optimal complexity. *Discrete & Computational Geometry*, 50(3):784–810, 2013.

[ABS13] Evmorfia N. Argyriou, Michael A. Bekos, and Antonios Symvonis. Maximizing the total resolution of graphs. *The Computer Journal*, 56(7):887–900, 2013.

[ADK$^+$15] Patrizio Angelini, Walter Didimo, Stephen Kobourov, Tamara Mchedlidze, Vincenzo Roselli, Antonios Symvonis, and Stephen Wismath. Monotone drawings of graphs with fixed embedding. *Algorithmica*, 71(2):233–257, 2015.

[AKV15] Muhammad Jawaherul Alam, Stephen G. Kobourov, and Sankar Veeramoni. Quantitative measures for cartogram generation techniques. In *Computer Graphics Forum*, volume 34, pages 351–360. Wiley Online Library, 2015.

[Ber99] François Bertault. A force-directed algorithm that preserves edge crossing properties. In *International Symposium on Graph Drawing*, pages 351–358. Springer, 1999.

[BHMvS19] Gregor Betz, Michael Hamann, Tamara Mchedlidze, and Sophie von Schmettow. Applying argumentation to structure and visualize multi-dimensional opinion spaces. *Argument & Computation*, 10(1):23–40, 2019.

[BKSB95] Thomas Brinkhoff, Hans-Peter Kriegel, Ralf Schneider, and Alexander Braun. Measuring the complexity of polygonal objects. In *ACM-GIS*, page 109. Citeseer, 1995.

[BP90] Karl-Friedrich Böhringer and Frances Newbery Paulisch. Using constraints to achieve stability in automatic graph layout algorithms. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 43–51, 1990.

[CS05] Yinpeng Chen and Hari Sundaram. Estimating complexity of 2d shapes. In *2005 IEEE 7th Workshop on Multimedia Signal Processing*, pages 1–4. IEEE, 2005.

[Dac19] Carsten Dachsbacher. Visualisierung. `https://cg.ivd.kit.edu/lehre/ss2019/index_2151.php`, 2019. Lecture at Karlsruhe Institute of Technology, summer term 2019.

[DFPP90] Hubert De Fraysseix, János Pach, and Richard Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990.

[DG02] Stephan Diehl and Carsten Görg. Graphs, they are changing. In *International Symposium on Graph Drawing*, pages 23–31. Springer, 2002.

[Ead84] Peter Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[ELMS91] Peter Eades, Wei Lai, Kazuo Misue, and Kozo Sugiyama. Preserving the mental map of a diagram. Technical report, Technical Report IIAS-RR-91-16E, Fujitsu Laboratories, 1991.

[Fel13] Stefan Felsner. Exploiting air-pressure to map floorplans on point sets. In *International Symposium on Graph Drawing*, pages 196–207. Springer, 2013.

[FGH74] Herbert J. Fleischner, Dennis P. Geller, and Frank Harary. Outerplanar graphs and weak duals. *The Journal of the Indian Mathematical Society*, 38(1-4):215–219, 1974.

[Fá48] István Fáry. On straight-line representation of planar graphs. *Acta scientiarum mathematicarum*, 11(229-233):2, 1948.

[GHK09] Emden R. Gansner, Yifan Hu, and Stephen G. Kobourov. GMap: Drawing graphs as maps. *arXiv preprint arXiv:0907.2585*, 2009.

[GN04] Michael T. Gastner and Mark E. J. Newman. Diffusion-based method for producing density-equalizing maps. *Proceedings of the National Academy of Sciences*, 101(20):7499–7504, 2004.

[ITK98] Tomonori Izumi, Atsushi Takahashi, and Yoji Kajitani. Air-pressure-model-based fast algorithms for general floorplan. In *Proceedings of 1998 Asia and South Pacific Design Automation Conference*, pages 563–570. IEEE, 1998.

[KKN13] Jan-Hinrich Kämper, Stephen G. Kobourov, and Martin Nöllenburg. Circular-arc cartograms. In *2013 IEEE Pacific Visualization Symposium (PacificVis)*, pages 1–8. IEEE, 2013.

[Kle18] Linda Kleist. Drawing planar graphs with prescribed face areas. *Journal of Computational Geometry*, 9(1):290–311, 2018.

[Kle19] Linda Kleist. *Planar graphs and face areas: Area-Universality.* Doctoral thesis, Technische Universität Berlin, Berlin, 2019.

[Kob13] Stephen G. Kobourov. Force-directed drawing algorithms. In Roberto Tamassia, editor, *Handbook of Graph Drawing and Visualization*, chapter 12, pages 383–408. CRC Press, 1^{st} edition, 2013.

[LLY06] Yi-Yi Lee, Chun-Cheng Lin, and Hsu-Chun Yen. Mental map preserving graph drawing using simulated annealing. In *Proceedings of the 2006 Asia-Pacific Symposium on Information Visualisation-Volume 60*, pages 179–188. Citeseer, 2006.

[MELS95] Kazuo Misue, Peter Eades, Wei Lai, and Kozo Sugiyama. Layout adjustment and the mental map. *Journal of Visual Languages & Computing*, 6(2):183–210, 1995.

[MKH11] Daisuke Mashima, Stephen G. Kobourov, and Yifan Hu. Visualizing dynamic data with maps. *IEEE Transactions on Visualization and Computer Graphics*, 18(9):1424–1437, 2011.

[NAK18] Sabrina Nusrat, Muhammad Jawaherul Alam, and Stephen G. Kobourov. Evaluating cartogram effectiveness. *IEEE Transactions on Visualization and Computer Graphics*, 24(2):1077–1090, 2018.

[NK16] Sabrina Nusrat and Stephen G. Kobourov. The state of the art in cartograms. In *Computer Graphics Forum*, volume 35, pages 619–642. Wiley Online Library, 2016.

[NPR12] Martin Nöllenburg, Roman Prutkin, and Ignaz Rutter. Edge-weighted contact representations of planar graphs. In *International Symposium on Graph Drawing*, pages 224–235. Springer, 2012.

[PHG06] Helen C. Purchase, Eve Hoggan, and Carsten Görg. How important is the "mental map"? — an empirical investigation of a dynamic graph layout algorithm. In *International Symposium on Graph Drawing*, pages 184–195. Springer, 2006.

[SAAB11] Paolo Simonetto, Daniel Archambault, David Auber, and Romain Bourqui. Impred: An improved force-directed algorithm that prevents nodes from crossing edges. In *Computer Graphics Forum*, volume 30, pages 1071–1080. Wiley Online Library, 2011.

[Sch90] Walter Schnyder. Embedding planar graphs on the grid. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 138–148, 1990.

[SF03] André Skupin and Sara I. Fabrikant. Spatialization methods: a cartographic research agenda for non-geographic information visualization. *Cartography and Geographic Information Science*, 30(2):99–119, 2003.

[SSK16] Bahador Saket, Carlos Scheidegger, and Stephen G. Kobourov. Comparing node-link and node-link-group visualizations from an enjoyment perspective. In *Computer Graphics Forum*, volume 35, pages 41–50. Wiley Online Library, 2016.

[SSKB14] Bahador Saket, Paolo Simonetto, Stephen G. Kobourov, and Katy Börner. Node, node-link, and node-link-group diagrams: An evaluation. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2231–2240, 2014.

[SSKB15] Bahador Saket, Carlos Scheidegger, Stephen G. Kobourov, and Katy Börner. Map-based visualizations increase recall accuracy of data. In *Computer Graphics Forum*, volume 34, pages 441–450. Wiley Online Library, 2015.

[Ste51] Sherman K. Stein. Convex maps. *Proceedings of the American Mathematical Society*, 2(3):464–466, 1951.

[Tho92] Carsten Thomassen. Plane cubic graphs with prescribed face areas. *Combinatorics, Probability and Computing*, 1(4):371–381, 1992.

[Tob04] Waldo Tobler. Thirty five years of computer cartograms. *ANNALS of the Association of American Geographers*, 94(1):58–73, 2004.

[Tut63] William Thomas Tutte. How to draw a graph. *Proceedings of the London Mathematical Society*, 3(1):743–767, 1963.

[vS17] Sophie von Schmettow. Visualizing opinion space — interactive geographic map representations for dynamic opinion datasets. Master's thesis, Karlsruhe Institute of Technology, 2017. `https://i11www.iti.kit.edu/_media/teaching/theses/sophie_final.pdf`.

[Wag36] Klaus Wagner. Bemerkungen zum vierfarbenproblem. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 46:26–32, 1936.

[Wag16] Dorothea Wagner. Algorithmen für planare graphen. `https://i11www.iti.kit.edu/teaching/sommer2016/planargraphs/index`, 2016. Lecture at Karlsruhe Institute of Technology.

[War19] Colin Ware. *Information visualization: perception for design.* Morgan Kaufmann, 2019.