Dynamic Graph Map Animation

Seok-Hee Hong[*] Peter Eades[†] Marnijati Torkel[‡] Weidong Huang[§] Cristina Cifuentes[¶]

Abstract

Recent methods for visualizing graphs have used a *map* metaphor: vertices are represented as regions in the plane, and proximity between regions represents edges between vertices.

In many real world applications, the data changes over time, resulting in a *dynamic map*. This paper introduces new methods for representing dynamic graphs with map animation. More specifically, we present three different animation methods: *MDSV* (Multidimensional scaling - Voronoi), *TV* (Tutte - Voronoi) and *TD* (Tutte - dual). These methods support operations such as addition and deletion of vertices and edges. Each of our methods uses a kind of *matrix interpolation*.

Index Terms: Human-centered computing—Visualization—Visualization techniques—Graph drawing; Human-centered computing—Visualization—Visualization design and evaluation methods;

1 Introduction

Recent methods for visualizing graphs have used a *map metaphor*: vertices are represented as simple regions in the plane, and proximity between regions represents similarity between vertices. An example is in Figure 1: here, files of Linux libraries are represented as “countries”, and functions within each file are sub-regions. Two “countries” are close to each other if the corresponding files are closely coupled in the software [4]. Similar methods have been used to visualize data from social networks, biological networks [5, 7].

In many real world applications, the graph changes over time, and is visualized as a *dynamic map*, as in Algorithm 1 below:

```
1  A graph $G_0$ is pictured on the screen as a map $D_0$.
2  The graph changes a little, resulting in a new graph $G_1$.
3  The system constructs a map $D_1$ representing $G_1$.
4  The system constructs and shows an “in-betweening”
     animation that visually transforms $D_0$ to $D_1$.
```

Algorithm 1: Dynamic Map Process

This paper focusses on the final step of constructing an animation from D_0 to D_1. A more formal description of this step is in Section 3.1 below. We introduce three methods for representing dynamic graph data with map animation: *MDS-Voronoi map animation*, *Tutte-Voronoi Map animation*, and *Tutte-Dual Map animation*. We consider two aspects of the quality of animations: *smoothness*, and *topological consistency*, which are described in Section 3.

In this paper, we assume that all graphs are edge-weighted.

[*]The University of Sydney, seokhee.hong@sydney.edu.au

[†]The University of Sydney, peter.eades@sydney.edu.au

[‡]The University of Sydney, mtor0581@sydney.edu.au

[§]University of Technology Sydney, Weidong.Huang@uts.edu.au

[¶]Oracle Labs Australia, cristina.cifuentes@oracle.com

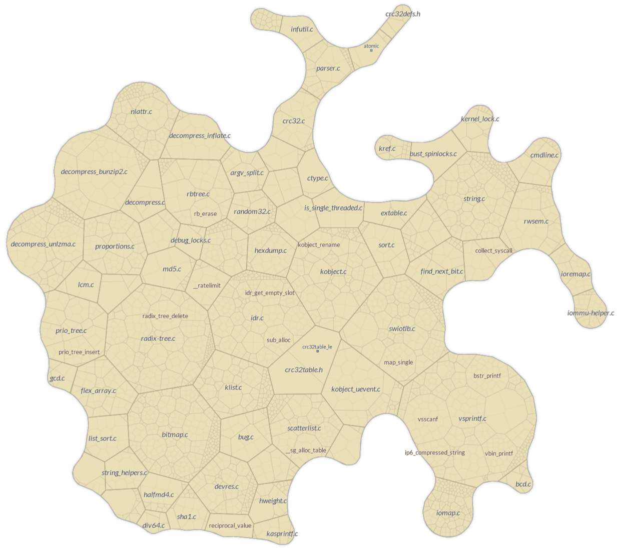


Figure 1: A map that represents an abstraction of Linux libraries [4].

2 Background

2.1 Preserving the mental map

In a dynamic visualization, the user must not lose the “mental map” of the screen as it changes [9]. Methods for preserving the mental map are available, including animation; see [1, 3].

2.2 Map representation of graphs

Map representations have been used to visualize graphs from a variety of applications. Hu et al. [5] introduced the *GMap* system, visualizing relational data with geographic-like maps. This representation shows structural information, clustering, and neighborhoods.

More specifically, they embed a (weighted) graph in two-dimensional space, analyze clusters and represent clusters as countries. The effectiveness of their approach has been demonstrated over a wide variety of domains, such as research collaboration networks, and maps of music styles and literary works; see [5]. Other methods, principally following the GMap approach, include *ConceptMap* [7] and *CodeMap* [4].

2.3 Dynamic graph visualization

Popular methods for visualizing dynamic graphs such as *small multiples*, *animation*, and *2.5D* visualization have been investigated extensively; see [1] for a survey. Most such work focuses on the classical node-link graph representation.

Those that deal with map representations concentrate on changing “overlays” of the map (the underlying map remains constant while graphical attributes such as colour change) [7, 8]. In contrast, this paper concentrates on changes to the underlying map.

2.4 Graph animation and morphing

An extensive literature on the topic of *graph morphing* has developed for node-link diagrams. The simplest method is linear interpolation: a drawing D_0 changes into a drawing D_1 of the same graph G with each vertex moving at constant speed along a line segment between its position in D_0 and its position in D_1.

Although linear interpolation is smooth and easy to implement, it is not good for preserving the mental map [3]. Newer methods concentrate on *poly-linear interpolation*, that is, contiguous sequences of linear interpolations [6].

3 The In-betweening Function, Smoothness, and Topological Consistency

3.1 The in-betweening function

Taking a formal mathematical viewpoint of Step 4 of the Dynamic Map Process in Algorithm 1, in-betweening consists of a function $D : [0,1] \to \mathscr{D}$, where $\mathscr{D}$ is the space of map representations, $D(0) = D_0$ and $D(1) = D_1$. Here $D(t)$ denotes the drawing at time t and is denoted by D_t.

The space $\mathscr{D}$ depends on the precise visualization metaphor that is used. For the map metaphor, each region of the map is a polygon, and we can take $\mathscr{D}$ to be the vector of polygons indexed by vertices. Note that map representation space $\mathscr{D}$ is a metric space (with a metric defined componentwise on the polygons). Thus mathematical notions of continuity and differentiability can be applied to in-betweening functions.

Note that in practice the mathematical in-betweening function D is discretised in Algorithm 2 as below. As $S \to \infty$, the discrete version approaches the abstract function D. In practice, the parameter S needs to be chosen just large enough to obtain a reasonable frame rate.

```
1 $\Delta t = 1/S$, where $S$ = number of steps for the animation;
2 $t = 0$;
3 while $t \leq 1$ do
4     Compute and render $D_t$ on the screen;
5     $t = t + \Delta t$;
  end
```

Algorithm 2: Discretised in-betweening function D

3.2 Smoothness

Intuitively, an animation is "smooth" when objects on the screen move without discontinuities. This can be precisely measured as the (mathematical) smoothness of the function D. If $k \geq 0$ then D *has (differentiability) class* C^k (or *is* C^k*-smooth*) if the derivatives $D', D'', \ldots, D^{(k)}$ with respect to t exist and are continuous. The higher the value of k, the more smooth the animation. At the limit, f is of class C^∞ (or just *smooth*) if it has derivatives of all orders.

3.3 Topological consistency

As noted in [9], one aspect of preserving the mental map is preserving "topology". Intuitively, this means that the adjacencies between objects on the screen should not change in the course of the animation. For a given map representation in $\mathscr{D}$, two regions are *adjacent* if they share a common boundary. Note that adjacency is a stronger notion than proximity. The set of regions and adjacency relationships between regions form a graph G. Note that G is a planar topological graph.

Two map representations D and D' are *topologically equivalent* to the extent that topological graphs G and G' so formed are the same. The in-betweening function D is *topologically consistent* in so far as D_t is topologically equivalent to D_0 for all $0 \leq t \leq 1$.

Note that 100% topological consistency is impossible in many cases; for example, if a new region is added to the map. For such cases, one could define a metric for topological consistency by measuring the approximate equivalence of the graphs G and G'; in this paper we do not give a formal definition of such a metric but leave it as an informal notion.

4 Dynamic Graph Map Animation

In Sections 4.1, 4.2, and 4.3 below, we define three methods for constructing a map from a graph, and, for each of these map construction methods, we define an animation method. We consider three atomic kinds of change for the graph, that is, three basic ways in which the graph G_0 changes to the graph G_1 in Step 2 of Algorithm 1.

1. *Edge-weight change*: G_0 and G_1 have the same vertices and edges but the edge weights differ.

2. *Vertex insertion/deletion*: G_1 is obtained from G_0 by insertion or deletion of a vertex (together with incident edges).

3. *Edge flips*: G_1 has the same vertices as G_0, but their edge sets differ by an *edge flip*, i.e., G_1 is formed by deleting an edge (u_0, v_0) and adding an edge (u_1, v_1), where u_0, v_0, u_1 and v_1 are distinct. This is mostly applied when the graph has a 4-cycle (u_0, u_1, v_0, v_1), and the edge flip essentially swaps chords on this 4-cycle. An edge flip operation is illustrated in Figure 2.

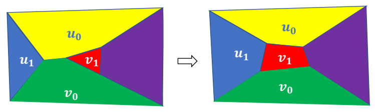


Figure 2: The edge flip operation for a map representation of the graph: adjacency between u_0 and v_0 is deleted, and u_1 becomes adjacent to v_1.

In practice, changes from G_0 to G_1 consist of a (fairly small) sequence of these atomic changes. In particular, note that the edge flip is commonly used to transform one graph into another; see [2].

Each method consists of (1) a *map creation algorithm* (step 3 of Algorithm1), and (2) *animation algorithms* (step 4 of Algorithm1); different algorithms for different changes at step 2 of Algorithm 1.

4.1 The MDS-Voronoi approach

For the first method, *map creation* is done using two steps:

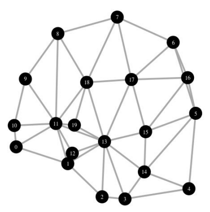
1. A *Multi Dimensional Scaling (MDS)* method computes a location $\Lambda(v)$ for each vertex v of G. The MDS layout uses a matrix M of "dissimilarities" between the vertices; for example, the entry M_{uv} of M is the (weighted) graph-theoretic distance between vertices u and v. MDS methods aim to ensure that the Euclidean distance $|\Lambda(u) - \Lambda(v)|$ between vertices u and v is approximately the same as the dissimilarity M_{uv} between the vertices u and v.

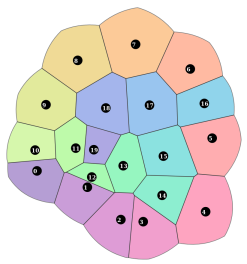
2. The map is then the Voronoi diagram of $\{\Lambda(v) : v \in G\}$.

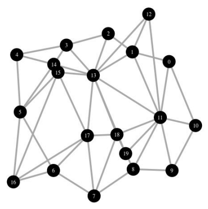
For a given dissimilarity matrix M, we denote the map obtained by this MDS-Voronoi approach by $MDSV(M)$. Figure 3 shows an example of an MDS-Voronoi map of a graph.

Note that a Voronoi diagram has some unbounded regions; there are two ways to avoid these in the map: (1) by "rounding" the outside boundary of the Voronoi regions on the convex hull, or (2) by creating a number of dummy vertices to make an outside boundary, but rendering the corresponding regions as invisible. The example in Figures 3 uses the first approach.

The *animation* algorithm for a *weight-change* operation uses *matrix interpolation* on the dissimilarity matrices, as follows. Suppose that M_0 and M_1 are the dissimilarity matrices used to compute maps D_0 and D_1 of graph G_0 and G_1. For a timestamp t with $0 \leq t \leq 1$, denote $(1-t)M_0 + tM_1$ by M_t. Then the map D_t at time t is the MDS-Voronoi map $MDSV(M_t)$.


(a) MDS layout Λ_0


(b) Voronoi map D_0 of Λ_0


(c) MDS layout Λ_1

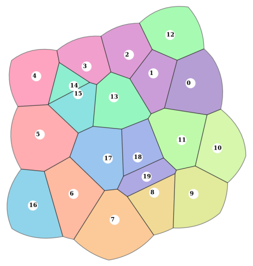
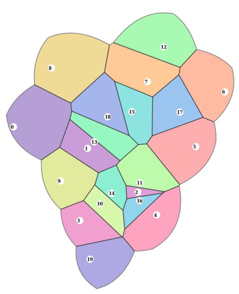
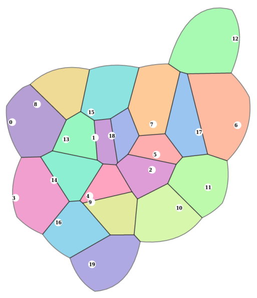
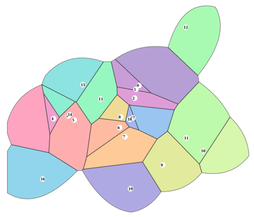
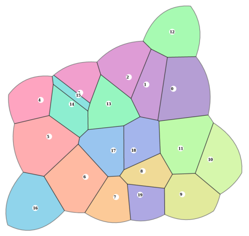

(d) Voronoi map D_1 of Λ_1

Figure 3: MDS-Voronoi maps.


(a)


(b)


(c)


(d)

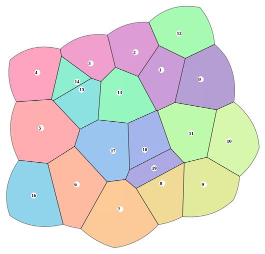

(e)

Figure 4: Screenshots of an MDS-Voronoi map animation for a weight-change operation. Although the animation is smooth, the topological changes are considerable.

This in-betweening function is discretised as in Algorithm 2. Figure 4 shows five screenshots from an animation using the MDS-Voronoi method.

MDSV smoothness: The smoothness of the MDSV in-betweening depends on two things: (a) the smoothness of the MDS function that computes Λ; and (b) the smoothness of the function that creates a Voronoi diagram from Λ.

A number of methods may be used to implement MDS; the simplest of these methods is a kind of linear projection of the high-dimensional dissimilarities into 2D; the function that computes the vertex locations at time t is a smooth (C^∞) function of t.

The smoothness of the creation of the Voronoi diagram is more complex, since vertices of the Voronoi diagram can appear and disappear at discrete timestamps. However, each Voronoi polygon changes smoothly with t; that is, for each vertex v of G, the function $t \rightarrow P(t,v)$ that gives the Voronoi polygon $P(t,v)$ of v at timestamp t is smooth in the following sense.

Suppose that $\Delta(p,p') \in R^+$ is a measure of the difference between two polygons p and p' (for example, $\Delta(p,p')$ is the area of the symmetric difference of p and p'). Let V_t be the Voronoi diagram at time t, and let p_t^v be a polygon corresponding to vertex v in V_t. Then for every $\delta > 0$ there is an $\varepsilon > 0$ such that each polygon $p_t^v \in V_t$, $\Delta(p_t^v, p_{t'}^v) < \varepsilon$ whenever $|t'-t| < \delta$. In this sense, the MDSV method is smooth.

MDSV topological consistency: The MDS layout method does not directly address topological properties. In practice, a small change in the graph often leads to a large change in the adjacencies between the Voronoi regions as in Fig. 4; topological consistency does not hold in practice.

4.2 The Tutte-Voronoi approach

For the second method, *map creation* is done using two steps:

1. We use the *Tutte algorithm* (a.k.a. the *barycentre algorithm*) [11] to construct a layout Λ of a graph G, as follows. A set A of *fixed vertices* is chosen and placed in convex position. Then each other vertex is placed at the weighted barycenter of its neighbour vertices.

   We can express this placement as a system of linear equations

   $$Lx = b \tag{1}$$

   where x is the $2 \times |V-A|$ vector of locations of vertices in $V-A$, b is a $2 \times |V-A|$ constant vector, and L is the submatrix of the weighted Laplacian of G whose rows and columns correspond to vertices in $V-A$. The matrix L, called the *Tutte matrix*, is invertible, and the layout Λ is defined by the solution x of equation 1.

2. The map is the Voronoi diagram of the vertex locations defined by Λ.

Note that for the set A in step (1) above, we can use the same methods as with MDSV to deal with the outer face; dummy vertices that are rendered invisible in the final drawing. We denote the map created from a graph G by this Tutte-Voronoi method as $TV(G)$.

The *animation* algorithm for a *weight-change* operation may be implemented using *matrix interpolation*, this time on the Tutte matrices. This method follows ideas from the *isotopy* literature [10].

Suppose that L_0 and L_1 are the Tutte matrices used to compute maps $TV(G_0) = D_0$ and $TV(G_1) = D_1$ of graphs G_0 and G_1. We assume that the fixed vertex set A is the same for both G_0 and G_1; if not, we can use a dummy fixed vertex set for each of G_0 and G_1.

For a timestamp t with $0 \le t \le 1$, denote $(1-t)L_0 + tL_1$ by L_t. Then the map D_t at time t is the Tutte-Voronoi map $TV(L_t)$. Again, this in-betweening function is discretised as in Algorithm 2.

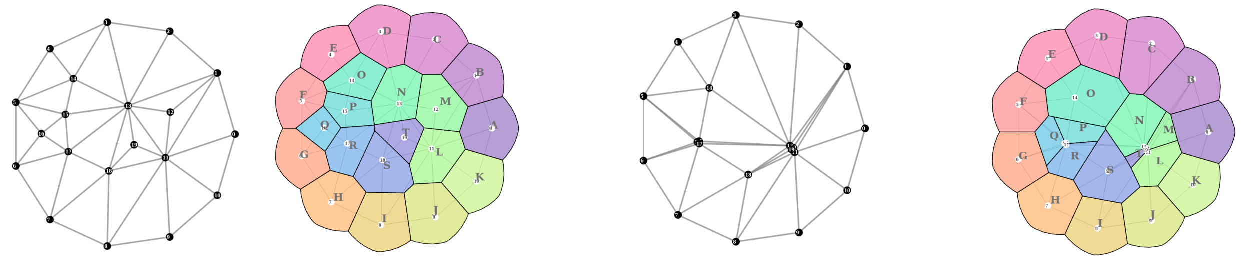

(a) Tutte layout Λ_0 (b) Voronoi map D_0 of Λ_0 (c) Tutte layout Λ_1 (d) Voronoi map D_1 of Λ_1

Figure 5: Tutte-Voronoi map; edge weights of a small number of edges are increased significantly, causing region T to almost disappear.

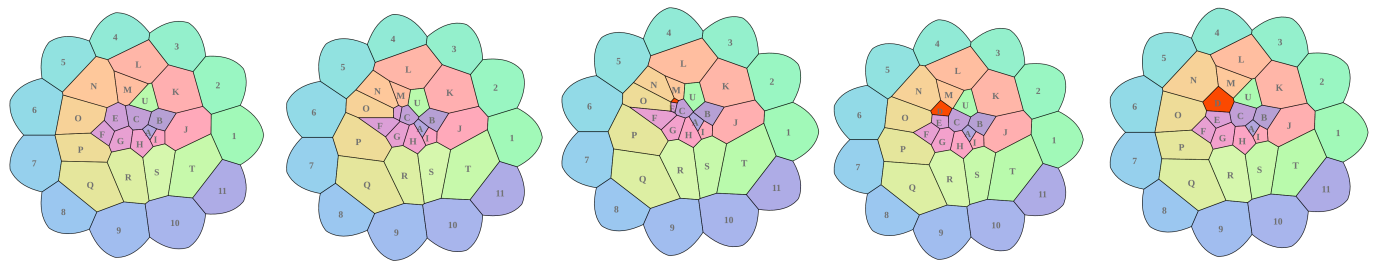
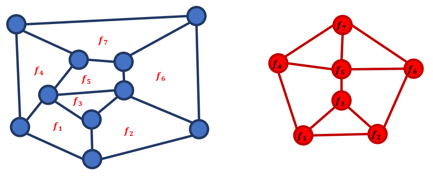

(a) (b) (c) (d) (e)

Figure 6: Screenshots for insertion on Tutte-Voronoi map animation. Vertex D is inserted.

Figure 5 shows examples of Tutte-Voronoi maps of graphs, with a weight-change operation.



(a) (b)

Figure 7: (a) A planar graph G and (b) its weak planar dual G'. Note that (a) is a map representation of (b).

The Tutte-Voronoi approach can be used for *vertex insertion/deletion*, with the following animation algorithm. Suppose that we insert a new vertex u into a graph G_0 to form a new graph G_1; u is adjacent to vertices $v_1, v_2, \ldots, v_k$ (the v_i are common to G_0 and G_1). We want the region of the map D_0 of G_0 corresponding to u to grow from a single point to be a convex polygon in the map D_1 of G_1.

Consider the graph G_0' formed from G_0 by inserting u; define $G_{0.5}$ to be same graph as G_0' but with very large weights on the all edges (u, v_i), $i = 1, 2, \ldots, k$. In the Tutte drawing $D_{0.5}$ of $G_{0.5}$, vertices $v_1, v_2, \ldots, v_k$ are very close together; in the corresponding Voronoi diagram, the region corresponding to u is very small.

Thus the animation from D_0 to D_1 proceeds as follows. Let L_0', $L_{0.5}$, and L_1 be the Tutte matrices corresponding to G_0', $G_{0.5}$, and G_1, and let

$$L_t = \begin{cases} (1-2t)L_0' + 2tL_{0.5} & : 0 < t < 0.5 \\ (1-2(t-0.5))L_{0.5} + 2(t-0.5)L_1 & : 0.5 < t < 1 \end{cases}$$

Figure 6 shows screenshots of the Tutte-Voronoi map insertion animations.

Deletion of a vertex essentially follows the reverse of the insertion process.

TV smoothness Note that the layout function $x = L_t^{-1} b$ is a non-linear but smooth (C^∞) function of t (because each entry in L_t^{-1} is a rational function in t, and the denominators are never zero). Furthermore, the Voronoi diagram is smooth, following the same reasoning as with MDSV above.

TV topological consistency From the Tutte Theorem [11], if G is planar, then the layout gives a planar topological graph. However, the Voronoi diagram step may result in topological inconsistency. In practice, the topological consistency of the Tutte-Voronoi method is better than that of the MDS-Voronoi method.

4.3 The Tutte-Dual approach

The third method also uses a Tutte drawing to compute layouts. However, instead of using Voronoi diagram map, this method uses a *weak planar dual map*.

Suppose that G is a planar graph embedded in the plane. The edges of G divide the plane into regions called *faces*. The *weak planar dual* of G is a graph G' with a vertex for every bounded face of G, and an edge between two bounded faces that share an edge of G.

An example of a planar graph and its weak planar dual is in Figure 7. Note that the primal graph in Figure 7(a) can be considered to be a map representation of the dual graph in Figure 7(b).

Suppose that G is a planar graph and G' is its (weak) planar dual. As long as G' is 3-connected, we can apply Tutte's algorithm to G' to get a map representation of G. If G' is not 3-connected, then we can augment by adding dummy edges.

Each edge in G' is dual to an edge in G; the weight of the dual edge is the same as the weight of the primal edge. Note that for the Tutte-Dual method, the Tutte algorithm is applied to the *weak planar dual* G' of G, not to G as in the Tutte-Voronoi method.

A *weight-change* operation can be implemented as follows. If D_0 and D_1 are Tutte-Dual representations of graphs G_0 and G_1 with

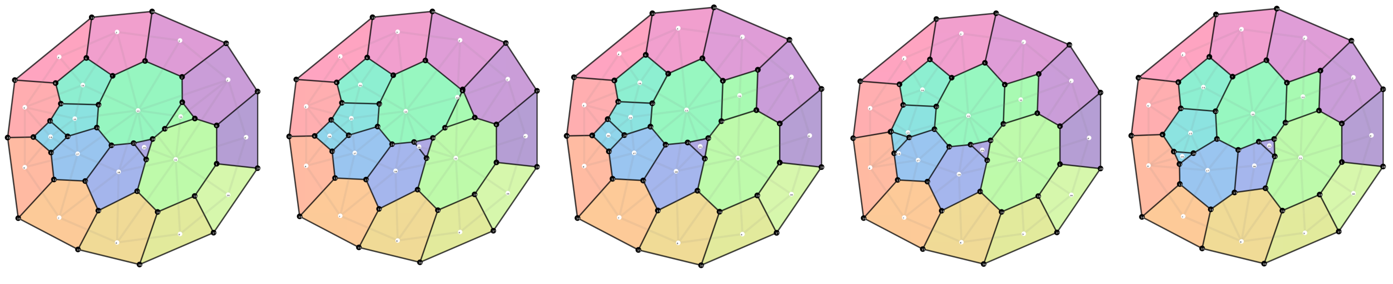

(a) (b) (c) (d) (e)

Figure 8: Screenshots for Tutte-Dual map animation for an edge-flip.

planar duals G'_0 and G'_1, then we can animate between D_0 and D_1 by matrix interpolation on the Tutte matrices L'_0 and L'_1 of G'_0 and G'_1, in the same way as for the Tutte-Voronoi method.

Vertex insertion and *deletion* are easily supported in this method. For deletion, the weights of the edges incident with a vertex u to be deleted are increased until they are very large. Thus the weights of the dual edges that bound the region corresponding to u become very large, and the region becomes very small. Eventually the region disappears. Insertion is the reverse of deletion.

Importantly, the Tutte-Dual method supports *edge flipping* operations. An edge flip transforms a graph G_0 to a graph G_1 with the same vertex set, as illustrated in Figure 2.

To perform the edge flip operation, we use *merge/split* operation on vertices, as follows. Suppose that the dual edge corresponding to (u_0, v_0) in G_0 is (a_0, b_0). First, the weight of the edge (a_0, b_0) is gradually increased to be very large; effectively this means that, in the primal, a_0 and b_0 become very close together.

Next, we merge vertices a_0 and b_0 to form a single vertex c; then c is split into two vertices a_1 and b_1, with an edge (a_1, b_1); this is the dual edge for the edge (u_1, v_1) in G_1. The edge (a_1, b_1) begins with very large weight so that a_1 and b_1 are very close together. Then we gradually decrease the weight of (a_1, b_1) to become the weight of (u_1, v_1). Figure 8 shows a number of steps of the Tutte Dual map edge flip animation.

TD smoothness The Tutte-Dual method is C^∞ smooth, following the same reasoning as for Tutte-Voronoi method.

TD topological consistency If the input graphs G_0 and G_1 are planar and have a common outer face, the Tutte-Dual method is topologically consistent. This follows from the Tutte's Theorem [11].

4.4 Summary and Discussion

This paper describes three dynamic graph map animation algorithms: MDS-Voronoi, the Tutte-Voronoi, and Tutte-dual. These methods support weight-changes,insertion and deletion of vertices, and edge flip operations in a dynamic graph.

From our (subjective) experience, we can summarize the pros and cons of each method are as follows. The take-away message of this paper is that the Tutte based animation methods produce better animations: they are smoother, and have superior topological consistency, and support more operations.

The MDS-Voronoi method smoothly supports weight-changes, and is suitable for multi-attribute graphs. However, in practice, it is far from topological consistent. The Tutte-Voronoi method smoothly supports weight changes and vertex insertion/deletion, and has better topological consistency than MDS-Voronoi. The Tutte-Dual method is better, on both smoothness and topological consistency, and it also supports edge-flip operations.

Our implementations use D3, Igraph, R and JavaScript; we used off-the-shelf routines for the underlying algorithms for MDS, Voronoi diagrams, and solving the Tutte equations. All three methods run in real time (less than 0.01 second per frame) for graphs of up to 100 vertices.

The theoretical worst-case time complexity depends on the underlying algorithms. It is clear that more sophisticated implementations are needed for large graphs; this is future work.

Other future work also includes a more systematic the evaluation. Human perception of the animations can be measured using HCI-style experiments. The quality metrics for smoothness and topological consistency can be refined.

Acknowledgement

This work was supported by an ARC (Australian Research Council) Linkage Grant with Oracle Research Lab.

References

[1] F. Beck, M. Burch, S. Diehl, and D. Weiskopf. A taxonomy and survey of dynamic graph visualization. *Comput. Graph. Forum*, 36(1):133–159, 2017. doi: 10.1111/cgf.12791

[2] F. Chung and R. Graham. Edge flipping in graphs. *Advances in Applied Mathematics*, 48:37–63, 2012.

[3] C. Friedrich and P. Eades. Graph drawing in motion. *J. Graph Algorithms Appl.*, 6(3):353–370, 2002. doi: 10.7155/jgaa.00057

[4] N. Hawes, S. Marshall, and C. Anslow. Codesurveyor: Mapping large-scale software to aid in code comprehension. In *3rd IEEE Working Conference on Software Visualization, VISSOFT 2015, Bremen, Germany, September 27-28, 2015*, pp. 96–105. IEEE Computer Society, 2015. doi: 10.1109/VISSOFT.2015.7332419

[5] Y. Hu, E. R. Gansner, and S. G. Kobourov. Visualizing graphs and clusters as maps. *IEEE Computer Graphics and Applications*, 30(6):54–66, 2010. doi: 10.1109/MCG.2010.101

[6] L. Kleist, B. Klemz, A. Lubiw, L. Schlipf, F. Staals, and D. Strash. Convexity-increasing morphs of planar graphs. In *Graph-Theoretic Concepts in Computer Science*, pp. 318–330, 2018.

[7] H. Liu, P. Eades, and S. Hong. Visualizing dynamic trajectories in social networks. In *2012 IEEE Symposium on Visual Languages and Human-Centric Computing*, pp. 197–204, 2012.

[8] D. Mashima, S. G. Kobourov, and Y. Hu. Visualizing dynamic data with maps. In G. Di Battista, J. Fekete, and H. Qu, eds., *IEEE Pacific Visualization Symposium, PacificVis 2011, Hong Kong, China, 1-4 March, 2011*, pp. 155–162. IEEE Computer Society, 2011. doi: 10.1109/PACIFICVIS.2011.5742385

[9] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *J. Vis. Lang. Comput.*, 6(2):183–210, 1995. doi: 10.1006/jvlc.1995.1010

[10] C. Thomassen. Deformations of plane graphs. *Journal of Combinatorial Theory B*, 34:244–257, 1983.

[11] W. T. Tutte. How to draw a graph. *Proceedings of the London Mathematical Society*, 13(3):743–767, 1963.