Drawing Knots with Tutte Embedding

Cameron Browne

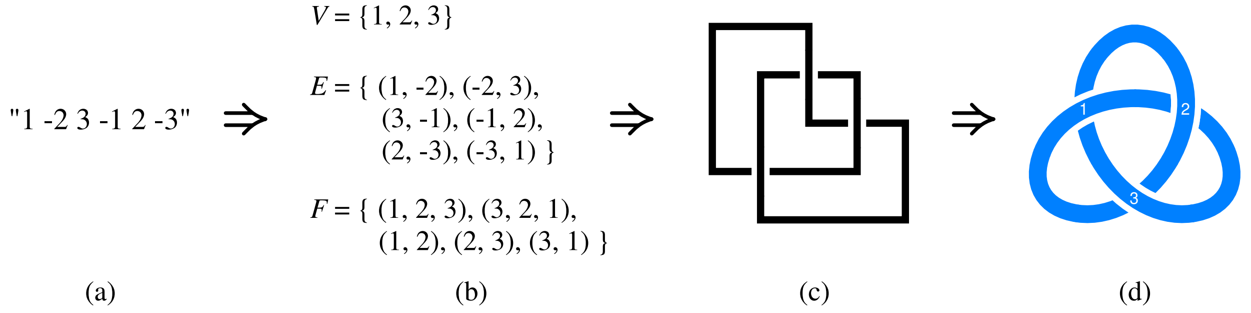
Maastricht University, The Netherlands; cameron.browne@maastrichtuniversity.nl

Abstract

When drawing planar representations of mathematical knots, it is challenging to find 2D embeddings that minimise the amount of smoothing required to produce aesthetically pleasing results. This paper describes an approach based on Tutte embedding (TE) that produces simple planar embeddings of knot projections that do not require excessive smoothing to produce good drawings. The approach is described with examples and notes on its implementation.

Drawing Knots

This paper addresses the challenge of drawing 2D projections of mathematical knots onto the plane. Figure 1 shows one way of achieving this, starting with the standard *Gauss code* description of a knot projection (a) which lists the crossings in the order in which the knot path visits them (with underpasses negated). From the Gauss code it is straightforward to extract a graph (b) of the knot consisting of a set of vertices V corresponding to the crossings, a set of edges E corresponding to the arcs between them, and a set of faces F corresponding to the 2D regions bounded by the arcs [9]. This *underlying graph* is planar, 4-valent and virtual, i.e. we know the relationships between the vertices but not their positions in the plane at this point.



"1 -2 3 -1 2 -3" $\Rightarrow$

$V = \{1, 2, 3\}$

$E = \{ (1, -2), (-2, 3), (3, -1), (-1, 2), (2, -3), (-3, 1) \}$

$F = \{ (1, 2, 3), (3, 2, 1), (1, 2), (2, 3), (3, 1) \}$

Figure 1: *One way to draw a knot: Gauss code ⇒ underlying graph ⇒ grid diagram ⇒ smoothed drawing.*

The more difficult step is to *embed* this graph such that its vertices and edges are assigned positions in the plane. Figure 1(c) shows the standard *grid diagram* approach in which the underlying graph is embedded in a 2D rectangular grid, e.g. using an *arc presentation* method [5]. The diagram is then typically smoothed using physics-based relaxation techniques [15] to give the final aesthetically pleasing *knot drawing* (d).

Knot drawings should exhibit smooth curves with good continuity and large crossing angles [10]. Grid diagrams are not good choices in this respect, e.g. the vertex and edge positions in the above grid diagram (c) do not closely match their positions in the final drawing (d) very well. This problem gets worse as knot size increases and it is known that the arc presentation approach can be problematic for more complex cases [1].

Other ways to embed the underlying graph include “bead and stick” drawings [15], Lombardi drawings [10], circle packings [14], knot mosaics [12] and user input [4], to name a few. Embeddings provided by these approaches can still require considerable smoothing to achieve a satisfactory drawing, which is a problem for interactive real-time applications as smoothing is the most computationally expensive part of the knot drawing process. This paper presents a simple method to produce 2D knot embeddings that require minimal smoothing for satisfactory drawings, based on a famous algorithm by graph theorist W. T. Tutte.

Tutte Embedding

Tutte embedding (TE) is a method of embedding a graph in the plane by selecting a face, embedding its vertices to form the corners of a convex bounding polygon, then solving a system of linear equations to embed the remaining (internal) vertices at the *barycentre* of their neighbours [16]. If the graph is *simple* (i.e. has no loops or multi-edges), *planar* (i.e. can be drawn without edge crossings) and *triconnected* (i.e. no two vertices [and their incident edges] can be removed to split the graph into connected components), then TE produces a unique planar straight-edge embedding whose faces are all convex.

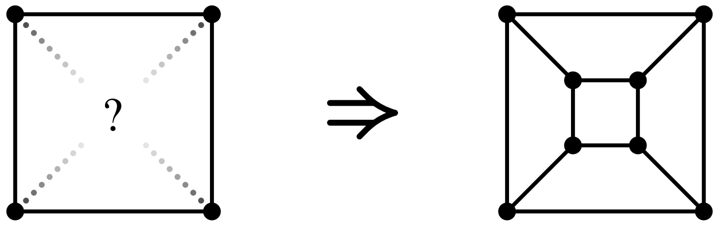


Figure 2: *Tutte Embedding (TE) of the graph of a cube projected onto the plane.*

Figure 2 shows the graph of a cube with $v = 8$ vertices embedded in the plane using TE. Any face of the cube is chosen as the bounding polygon and its $s = 4$ vertices embedded to form a square (left). The positions of the remaining (internal) vertices are not known yet. A system of $2(v - s) = 8$ linear equations is then set up to average the positions of the neighbours of the $v - s = 4$ internal vertices and solving this system of equations, e.g. by Gaussian elimination, yields their unique x and y positions in the plane (right).

Unfortunately, knot graphs tend to be biconnected rather than triconnected. Scharein solved this problem by replacing each vertex in the graph by a structure of four connected vertices to give triconnected "quad graphs", but found that TE produced poor embeddings for such graphs, with vertices clustered in small regions [15, p.102]. The following sections describe how TE can indeed produce useful knot embeddings.

Planar Triconnectivity

Planar triconnectivity is required for TE but few knots have projections that produce a triconnected underlying graph. Consider the *lens* shape shown in Figure 3 (left), consisting of a region bounded by two arcs, also called a *2-face*, *lune* or *bigon*. This shape is found abundantly in knots [6] and produces multi-edges in the resulting underlying graph if straight edges are used (Figure 3, middle).

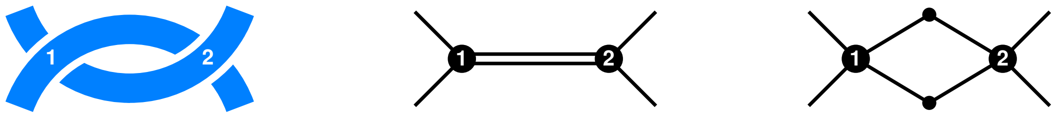


Figure 3: *Lenses produce straight multi-edges unless intermediate vertices are introduced.*

An obvious solution is to split multi-edges with intermediate vertices to produce a simple graph (Figure 3, right). We now distinguish between *crossing vertices* that correspond to crossings in the knot (large dots with labels) and *intermediate vertices* inserted between them (small dots). Unfortunately, the crossing vertices 1 and 2 now constitute a *separation pair* whose removal would disconnect the intermediate vertices, thus violating the triconnectivity requirement for TE to be successfully applied.

Figure 4(a) shows a more complex example that demonstrates why triconnectivity is required. Crossing vertices 1 and 4 constitute a separation pair whose removal would disconnect the other vertices contained in the shaded area (b). Further, within that disconnected set crossing vertices 2 and 3 constitute a nested separation pair whose removal would disconnect the two intermediate vertices.

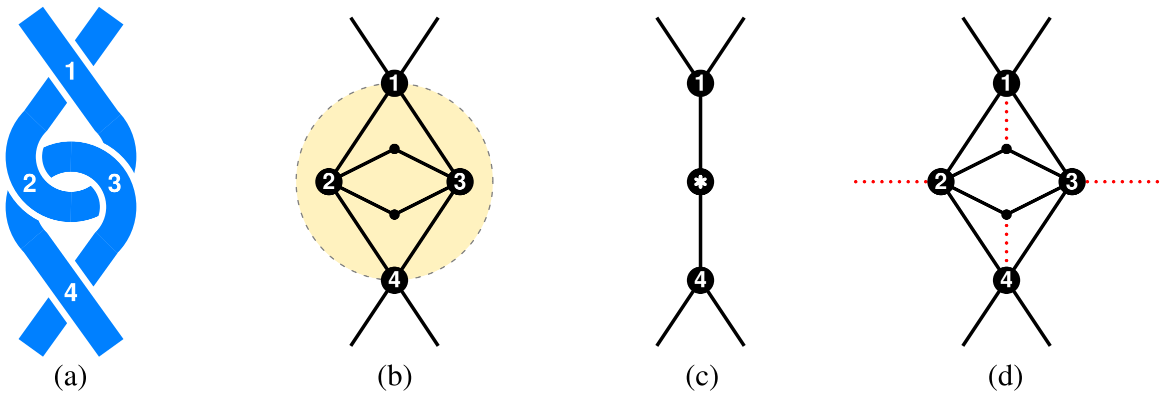


Figure 4: *Vertices 1 and 4 form a separation pair whose disconnected set will collapse unless augmented.*

When TE is performed on this structure, the positions of vertices 1 and 4 provide the only frame of reference for their disconnected set as all connections to the rest of the graph pass through them. The vertices in the disconnected set will therefore collapse to the line passing through their separation pair 1 and 4 when the TE barycentre calculation is applied, as indicated by the '*' in Figure 4(c).

However, temporary *virtual edges* can be strategically added to augment the graph and restore triconnectivity. Figure 4(d) shows virtual edges (dotted lines) added to connect each of the crossing vertices 2 and 3, as well as the two intermediate vertices, to some other vertex across their enclosing faces. Virtual edges connect potentially disconnected vertices to the rest of the graph, achieving triconnectivity and providing a broader frame of reference outside their separation pair for meaningful barycentre calculation. The question is *where* to insert such virtual edges in a given graph.

This task is similar to the *Planar Triconnectivity Augmentation* (PTA) problem of adding the minimum number of extra edges to a biconnected graph that will make it triconnected, for which algorithms that approximate $O(n^3)$ are known [8]. Savings can be made with knot graphs as we know the type of graph we are dealing with (i.e. simple, planar, biconnected and 4-valent) and there is no need in this case to find the minimum number of triconnecting edges. In fact, more edges in the graph allow more uniform embeddings, and hence better knot drawings, at negligible cost.[1]

Separation Pairs, Sets and Groups

It is useful to now define some terms related to planar triconnectivity. A *separation pair* is a pair of vertices (v_a, v_b) whose removal from the graph (along with any incident edges) would split the graph into more than one connected component. We choose the component with the fewest perimeter vertices, or if they have the same number then the component with the fewest vertices, and call its vertices the *disconnected set*.[2] Each separation pair and disconnected set together constitute a *separation set*, which can be referenced by its separation pair alone as the disconnected set is implied.

It is common to find chains of linked separation sets that form *separation groups* in which each member can reach all other members by steps through shared separation pair vertices. For each such group, a single *representative separation set* is identified which has the largest disconnected set in its group and includes the vertices of all other members. For example, the structure shown in Figure 5 consists of two separation groups $[(1,6),(1,5),(2,6),(2,5),(1,2),(5,6)]$ and $[(3,4)]$ with representatives (1, 6) and (3, 4), respectively. These are the separation sets that must be augmented with virtual edges to achieve triconnectivity.

[1]TE time complexity is based on the number of non-embedded vertices in the graph, not edges.

[2]If both counts match for both candidate vertex sets then either can be chosen, e.g. the set containing the lowest vertex index.

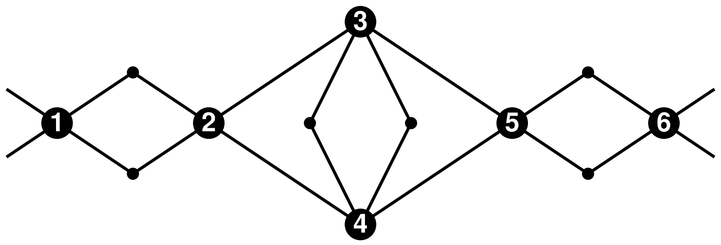


Figure 5: *This structure of linked separations has two separation groups represented by (1, 6) and (3, 4).*

Graph Augmentation

The graph augmentation strategy is simple: for each face F that partially (but not completely) contains the vertices of any representative separation set S, add at least one virtual edge connecting the disconnected set of S to a vertex of F outside S. For each separation set S incident with face F, we identify an *urgent set* of vertices that must be connected across F; these are the vertices in the disconnected set of S that are incident with F excluding crossing vertices adjacent to any intermediate vertex.[3] There are two cases to consider:

(a) ***Single Incident Set:*** If face F has exactly one incident separation set S, then nominate the furthest vertex (or two) around the face from any vertex in S as the *attachment vertices* and add a virtual edge from each urgent vertex in S to its closest attachment vertex.[4] If either of two equally far attachment vertices is an intermediate vertex, then only use that one. This is shown in faces F_a and F_b in Figure 6.

(b) ***Multiple Incident Sets:*** If face F has two or more incident separation sets S_1, S_2, ..., S_n, then connect consecutive sets by adding virtual edges between the last half of the urgent set of each S_i with the first half of the urgent set of the next consecutive S_{i+1} around F. If any S_i has only one urgent vertex then it constitutes both halves. This is shown in Figure 6 in faces F_c (two incident sets) and F_d (three incident sets).

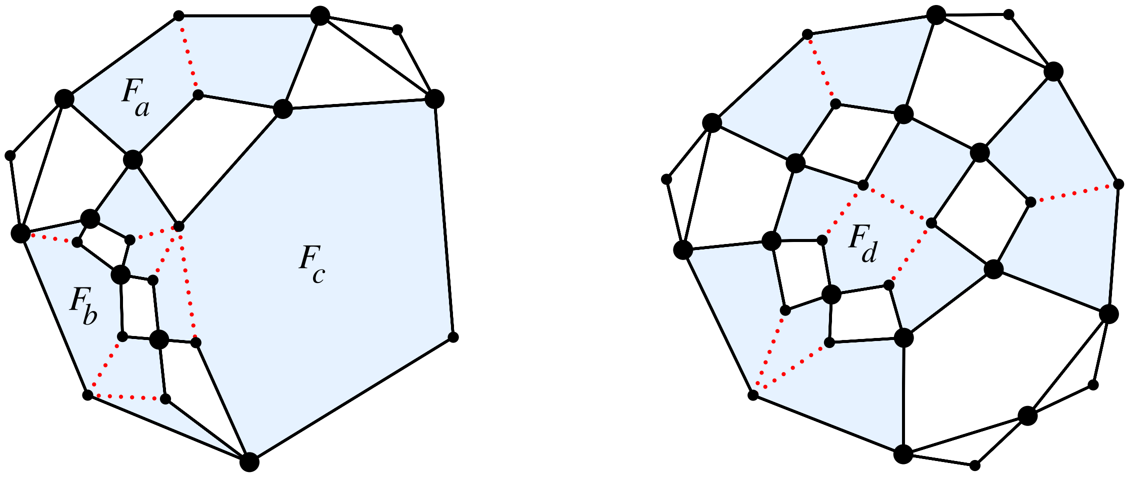


Figure 6: *Virtual edges (dotted lines) augment these graphs to achieve internal triconnectivity.*

Graphs thus augmented may contain perimeter vertices that are not triconnected, such as the graphs shown in Figure 6. But these perimeter vertices form the bounding polygon so have already been embedded; it is only the *internal* vertices yet to be embedded by TE that really need to be triconnected.

[3]Internal intermediate vertices come from lenses so are always more urgent to connect and dominate adjacent crossing vertices, e.g. intermediate vertices dominate adjacent crossing vertices for the virtual edge placements in faces F_b, F_c and F_d in Figure 6.

[4]Where "furthest" and "closest" here refer to the minimum number of edges around F between the two vertices.

Algorithm

This section describes the complete algorithm using knot projection 13ah_1131 from the Regina database [3] as an example. Figure 7 shows the smoothed drawing of this projection that we ultimately want to achieve, and Figure 8 shows each step of the algorithm as it is applied.

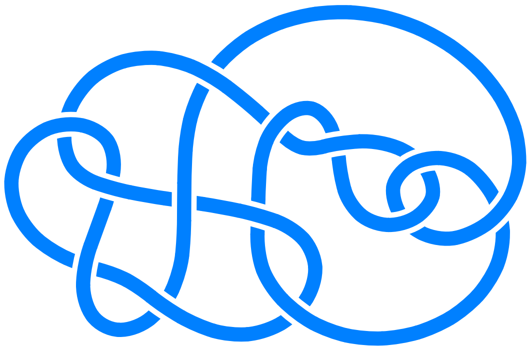


Figure 7: *Smoothed drawing of Regina knot 13ah_1131.*

(a) ***Bounding Polygon*****:** Select as perimeter a face F from the underlying graph with the most crossings; this yields better embeddings and "nicer" drawings [2]. If more than one such face, choose the maximal face with least most distant vertex. Embed the perimeter crossings around a circle to form the bounding polygon, spaced proportionately to the number of crossings in each outer face.[5] Zeng *et al.* [17] also embed the bounding polygon to a circle when applying TE for human face visualisation, but use uniform spacing.

(b) ***Perimeter Splitting*****:** Add an intermediate vertex at the midpoint between consecutive pairs of perimeter crossing vertices and embed them at the closest point on the boundary circle. This increases the embedding area, allows more options for adding virtual edges and generally yields a better spread of embedded vertices. This also converts perimeter lenses from multi-edges to single edge structures (lower left in Figure 8(b)).

(c) ***Augmentation*****:** Find all separation sets within the graph and discard those with three or more vertices on the graph perimeter.[6] Form separation groups, identify their representative separation sets, and discard the rest. For each face F, if F is incident with the disconnected set of any remaining separation set S and S does not cover all vertices in F, then find the urgent sets of all remaining separation sets incident with F (removing duplicates or urgent sets covered by other urgent sets). Augment the graph by adding virtual edges to connect the urgent sets across F according to the rules above (red dotted lines in Figure 8). Perform a TE pass.

(d) ***Triangulation*****:** Triangulate the graph by adding virtual edges to each non-triangular face (green dotted lines in Figure 8). Repeatedly clip each such face's most acute ear until it has four remaining vertices, then add the edge that divides it most evenly. Triangulation guarantees full triconnectivity and encourages the embedded vertices to spread more evenly at negligible cost. Perform a second TE pass.[7]

(e) ***Edge Removal*****:** Remove all virtual edges to give the final embedding (still with intermediate vertices).

(f) ***Drawing*****:** Draw the knot, e.g. by using the knot path vertices as the control points of a cubic B-Spline curve and redrawing overpasses. The results tend to be eccentric and not good drawings in themselves, but close to the final aesthetically pleasing shapes that can be achieved with minimal smoothing (compare Figure 8(f) with Figure 7). The smoothing algorithm implemented for this paper strives to match consecutive edge lengths and corner angles along the knot path while repelling non-adjacent graph elements.

[5]Proportionate spacing around the perimeter provides greater area for larger faces that will eventually contain more virtual edges.
[6]Because perimeter vertices form the bounding polygon and have already been successfully embedded in step (a).
[7]Lloyd's algorithm [11] may be applied at this point to spread the internal vertices even more evenly.

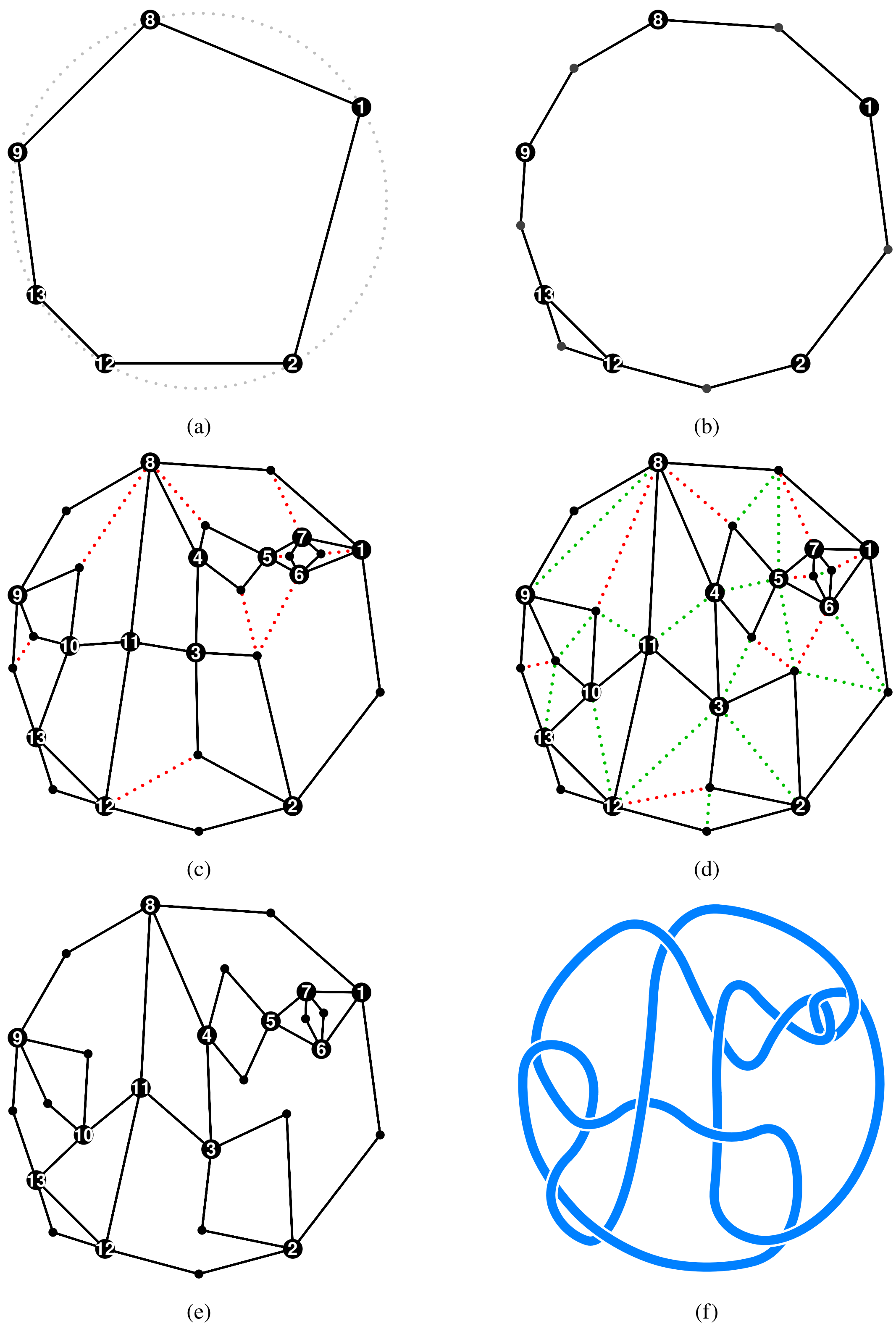


Figure 8: *Embedding 13ah_1131: (a) bounding polygon, (b) perimeter splitting, (c) TE after augmentation, (d) TE after triangulation, (e) virtual edges removed, and (f) drawn without smoothing.*

Performance

The algorithm achieves interactive speeds and embeds the 1,701,935 knots in the Regina database up to 16 crossings [3] at an average rate of ≈0.0005s per knot on a single thread of an Apple M1 chip. Computation time increases with the number of crossings, for example the 138-crossing projection of the unknot shown in Figure 9 (from [13, p. 19]) takes ≈0.1s to embed and smoothing it takes several seconds.

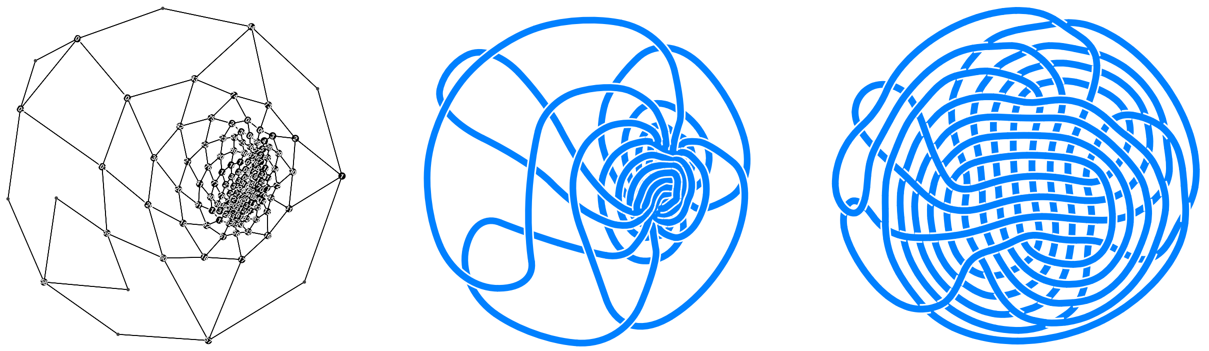


Figure 9: *138-crossing projection of the unknot embedded (left), drawn (middle) and smoothed (right).*

The algorithm's speed is limited by the efficiency of: 1) finding separation pairs, and 2) solving the system of TE barycentre equations (twice). Linear time algorithms exist for finding separation pairs within a graph (e.g. [7]) but a simpler brute force search can actually be faster for smaller knots.

Knots with twists (e.g. Figure 10, left) pose a problem for the algorithm, as each twist would give a duplicate separation pair (v_i, v_i). Such cases can be handled by removing twists from the Gauss code (by recursively removing adjacent occurrences of the same crossing) then inserting each twist back into the embedded graph by adding a crossing vertex and two or three intermediate vertices at the appropriate point.

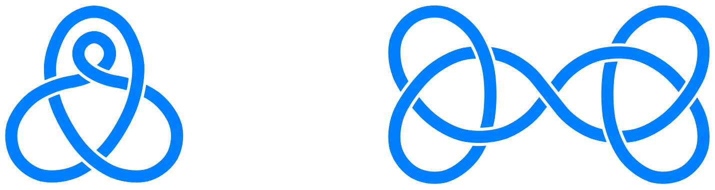


Figure 10: *Projections with twists (left) or perimeter bottlenecks (right) require special handling.*

Knots with bottlenecks or *articulation points* at which the perimeter visits the same crossing twice (e.g. Figure 10, right), also pose a problem, as their bounding polygon would not be convex. This can be solved by splitting the Gauss code into two sub-knots delineated by the bottleneck, embedding each, then splicing the two graphs together through a new crossing vertex.

Iterated Tutte Embedding

Separation pair testing can be avoided entirely by exploiting the side-effect that separation sets collapse to the line between their separation pair when TE is applied (Figure 4, c). It is then a simple matter to check for zero-area faces, attach virtual edges to them, and re-apply TE until no more such faces are found.

This process of *iterated Tutte embedding* (ITE) produces similar results to the algorithm outlined above but can be faster (depending on knot size and implementation) as explicit separation pair detection is not required. Furthermore, ITE provides additional geometric information with each iteration as more vertices are embedded, allowing more informed decisions regarding choice of target vertices for subsequent virtual edges. ITE seems to be a good choice for larger knots.

Conclusion

Tutte embedding allows a simple approach for embedding knot projections in the plane from their underlying graphs. The approach is robust, fast and yields simple embeddings that do not require excessive smoothing to produce aesthetically pleasing drawings. Future work will include comparisons between the iterated and non-iterated versions of the approach. Comparisons might also be made with other methods of embedding and drawing knots, in terms of both speed and the quality of their results.

Acknowledgements

This work was supported by the European Research Council as part of the *Digital Ludeme Project* (ERC Consolidator Grant #771292). Thanks to Frederic Maire for proofreading and useful suggestions.

References

[1] E. Beltrami and P. R. Cromwell. "A Limitation on Algorithms for Constructing Minimal Arc-Presentations from Link Diagrams." *Journal of Knot Theory and its Ramifications*, vol. 7, no. 4, 1998, pp. 415–423.

[2] C. Browne. "Nice Knots." *Bridges Proceedings*, Aalto, Finland, Aug. 1–5, 2022, pp. 309–312.

[3] B. A. Burton, R. Budney, W. Pettersson, *et al.* "Regina: Software for Low-Dimensional Topology." *GitHub*, version 7.0, 1999–2021. http://regina-normal.github.io/

[4] G. Costagliola, M. De Rosa, *et al.* "KnotSketch: A Tool for Knot Diagram Sketching, Encoding and Re-Generation." *Journal of Visual Languages and Sentient Systems*, vol. 2, 2016, pp. 16–25.

[5] P. C. Cromwell. *Knots and Links*. Cambridge University Press, 2004.

[6] S. Eliahou, F. Harary and L. Kauffman. "Lune-Free Knot Graphs." *Journal of Knot Theory and its Ramifications*, vol. 17, no. 1, 2008, pp. 55–74.

[7] C. Gutwenger and P. Mutzel. "A Linear Time Implementation of SPQR-Trees." *Graph Drawing (GD '00)*, Williamsburg, USA, 2000, pp. 77–90.

[8] G. Kant and H. L. Bodlaender. *Planar Graph Augmentation Problems*, Technical report, RUU-CS-91-25, Utrecht University, 1991.

[9] L. H. Kauffman. "Gauss Codes, Quantum Groups and Ribbon Hopf Algebras." *Reviews in Mathematical Physics*, vol. 5, no. 4, 1993, pp. 735–773.

[10] P. Kindermann, S. Kobourov, M. Löffler, *et al.* "Lombardi Drawings of Knots and Links." In *Graph Drawing and Network Visualization*, edited by F. Frati and K. L. Ma, Springer, 2018, pp. 113–126.

[11] S. P. Lloyd. "Least Squares Quantization in PCM." *IEEE Transactions on Information Theory*, vol. 28, no. 2, 1982, pp. 129–137.

[12] S. J. Lomanaco and L. H. Kauffman. "Quantum Knots and Mosaics." *Quantum Information Processing*, vol. 7, no. 2–3, 2008, pp. 85–115.

[13] C. Petronio and A. Zanellati. "Algorithmic Simplification of Knot Diagrams: New Moves and Experiments." *Journal of Knot Theory and Its Ramifications*, vol. 25, no. 10, 2016, pp. 1–30.

[14] P. Rideout. "Generating SVG for the Prime Knots." *The Little Grasshopper*. 2020. https://prideout.net/blog/svg_knots/

[15] R. G. Scharein. *Interactive Topological Drawing*, PhD thesis, University of British Columbia, 1998.

[16] W. T. Tutte. "How to Draw a Graph." *Proceedings of the London Mathematical Society*, vol. 3, no. 13, 1963, pp. 743–768.

[17] W. Zeng, Y.-J. Yang and M. Razib. "Graph-Constrained Surface Registration Based on Tutte Embedding." *IEEE CVPRW 2016*, Las Vegas, USA, 2016, pp. 516–523.