Iterated Tutte Embedding

Supplement to “Drawing Knots with Tutte Embedding”

Cameron Browne

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

Abstract

The paper “Drawing Knots with Tutte Embedding” [2] describes a method for producing planar embeddings of mathematical knots, by augmenting their knot graphs with temporary virtual edges to achieve triconnectivity then performing *Tutte embedding* (TE) to position the vertices. This supplementary paper describes an extension of this method called *Iterated Tutte Embedding* (ITE) that provides better results more efficiently (but with less theoretical justification). Please refer to the original paper for definitions of the terms used here.

Drawing Knots with Tutte Embedding

When drawing mathematical knots, *Tutte embedding* (TE) [4] is an obvious choice for embedding the vertices of the underlying graph in the plane to produce an initial knot diagram for smoothing. However, there is a problem in that TE assumes triconnectivity but this cannot be guaranteed for the underlying graphs of knots. For example, consider the structure shown in Figure 1 (from [2]).

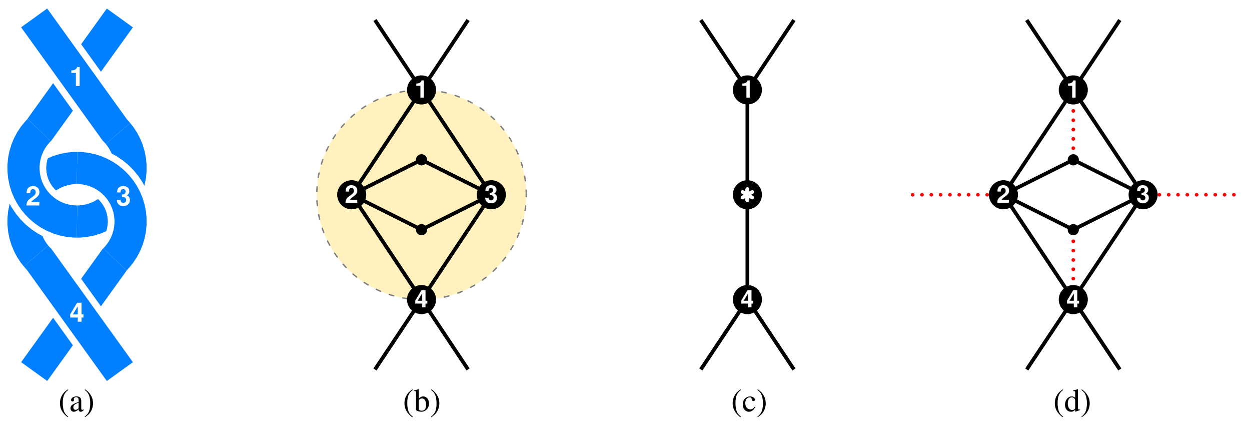


Figure 1: *Discconnected sets (b) collapse to the line between their separation pair (c) unless augmented (d).*

Crossing vertices 1 and 4 represent a *separation pair* whose removal would disconnect the shaded *disconnected set* (b). Applying TE would then collapse the vertices in the disconnected set to the line between 1 and 4, as these are the set’s only frame of reference (c). To correct this, we must *augment* the graph with temporary *virtual edges* to further connect the separated set and achieve triconnectivity (dotted lines, d).

Note that TE will always collapse such separation sets to “flat” faces with zero area.[1] We can exploit this fact to iteratively detect such cases and augment them with multiple TE passes. This is abusing TE somewhat – it was never intended for this purpose! – but the process works well in practice.

[1]In order to handle floating point (im)precision, a face is deemed to be *flat* if the square root of its absolute area is less than 0.01% of its average edge length.

Iterated Tutte Embedding

The *Iterated Tutte Embedding* (ITE) algorithm can be summarised as follows:

1. Choose a face F_p from the knot's underlying graph to be the diagram's perimeter.
2. Embed the vertices of F_p in a bounding circle.
3. Repeat until no flat faces are detected:
   a) Perform TE on all vertices and edges.
   b) For each flat face F_f, add virtual edges connecting it across each coincident non-flat face F_n.
4. Triangulate the diagram and perform a final TE pass.
5. Remove virtual edges.

Wherever a non-flat face F_n contains a sequence of vertices also present in a flat face F_f, that sequence will represent part of a *separation set*[2] in the current (partially augmented) graph. The first and last vertices of this sequence will be the separation pair and can be ignored. Similarly, any *crossing vertices* in the sequence adjacent to *intermediate vertices*[3] can also be ignored, as the intermediate vertices are more urgent to connect (see the original paper [2] for details). The remaining vertices in this sequence constitute the *urgent set* of vertices that must be connected across F_n with one or more virtual edges.

For example, Figure 2 shows a flat face F_f with vertex sequence (2, *, 1) shared by a non-flat face F_n with vertices (1, 3, 4, 5, 2, *). Ignoring the first and last vertices of this shared sequence leaves an urgent set consisting of vertex * that must be connected across F_n.

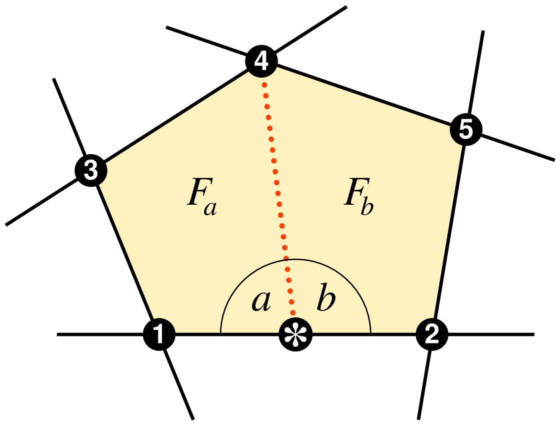


Figure 2: *Connecting urgent vertex * across an incident non-flat face with a virtual edge.*

For each urgent vertex in the urgent set, we must choose at least one target vertex in face F_n to connect to. We choose the target vertex (outside the shared vertex sequence) that is:

1. most perpendicular to the separation pair line (angles a and b are most similar), and
2. would split F_n most evenly into sub-faces F_a and F_b of similar area.

For example, this procedure would choose to add a virtual edge between urgent vertex $*$ and target vertex 4 in the case shown in Figure 2. This process is applied greedily, choosing the most preferred target vertex each time, until no more such virtual edges can be placed without intersection. In the case of a non-flat face F_n containing more than one urgent set, these are fully inter-connected without intersection, as described in the original paper [2]. Both cases are shown in the following walk-through.

[2]A *separation set* consists of a *separation pair* and its resulting *disconnected set*.
[3]An *intermediate vertex* is any vertex that does not correspond to a crossing in the knot.

Algorithm

This section describes the ITE algorithm by example, using knot 13ah_1131 from the Regina database [3] as per the original paper [2], for comparison. Figure 3 shows the smoothed drawing of this knot that we ultimately want to achieve, while Figures 4 and 5 show the steps of the algorithm as it is applied.

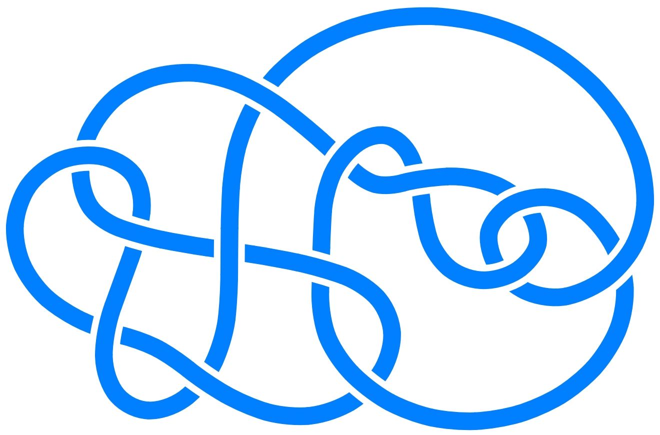


Figure 3: *Smoothed drawing of Regina knot 13ah_1131.*

(a) ***Bounding Polygon*****:** Choose a face F_p from the underlying graph to be the perimeter and embed its crossings around a circle to form the bounding polygon. The face with the most crossings is typically a good choice [1] but it can be preferable for ITE to choose the face that would maximise the number of perimeter lenses, in order to minimise the number of TE passes (perimeter lenses are embedded up-front in step (b)).

(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 embeds perimeter lenses, e.g. lower left in Figure 4 (b).

(c) ***TE Pass 1*****:** Perform a TE pass and check for flat faces.

(d) ***Augmentation Step 1*****:** For each non-flat face F_n incident with one or more flat faces F_f, add virtual edges to connect each urgent vertex across the face, as described in the previous section.

(e) ***TE Pass 2*****:** Perform another TE pass and check for flat faces.

(f) ***Augmentation Step 2*****:** For each non-flat face F_n incident with one or more flat faces F_f, add virtual edges to connect each urgent vertex across the face, as described in the previous section.

(g) ***TE Pass 3*****:** Perform another TE pass. No more flat faces are found so the iteration stops.

(h) ***Triangulation*****:** Triangulate the graph by adding virtual edges to each non-triangular face (green dotted lines in Figure 5). Repeatedly add the edge that divides the face most evenly until only triangles remain.

(i) ***Final TE Pass*****:** Perform a final TE pass to incorporate the triangulation.

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

(k) ***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. Note that this drawing is already visually smoother than that produced by the base TE method described in [2].

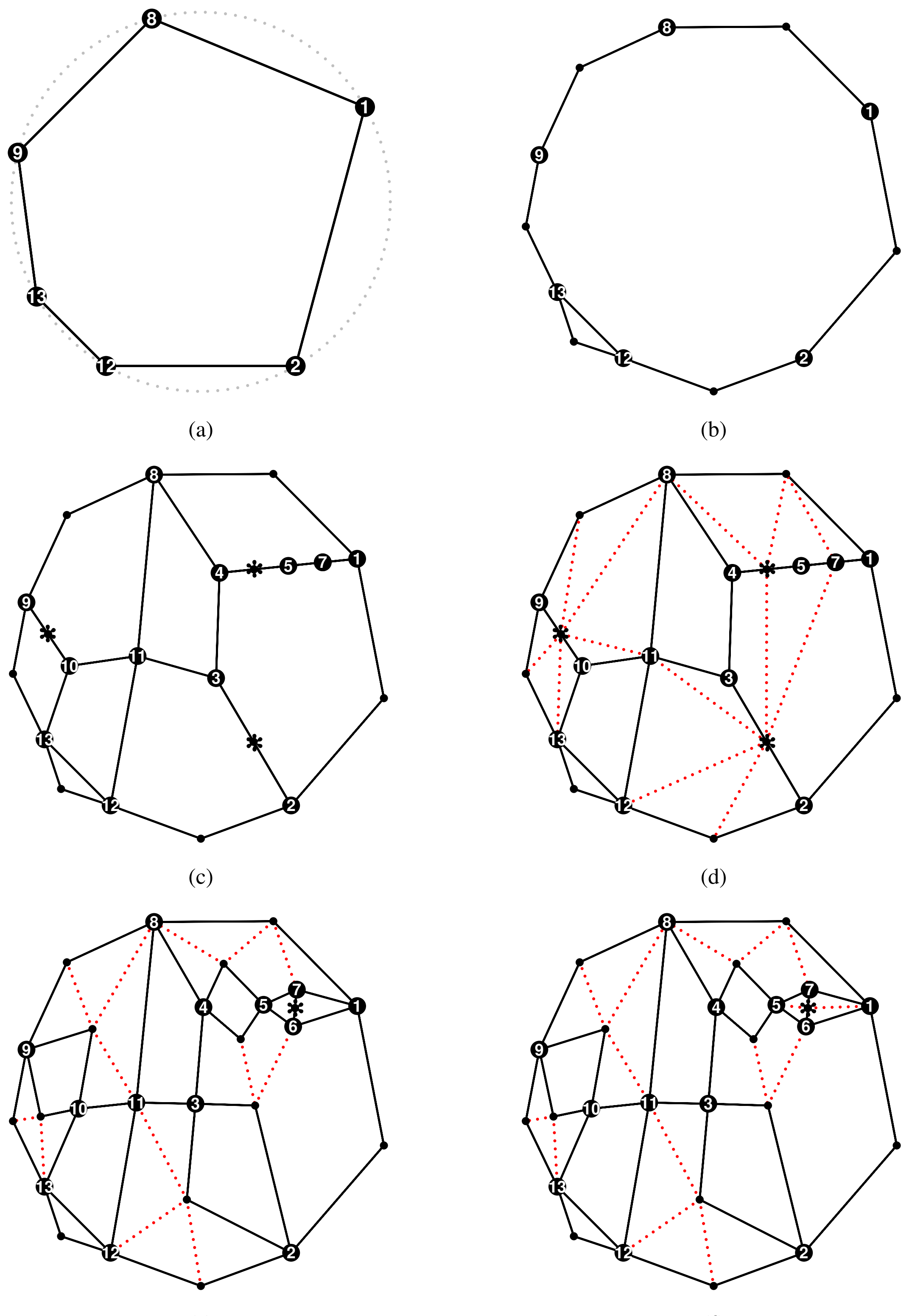
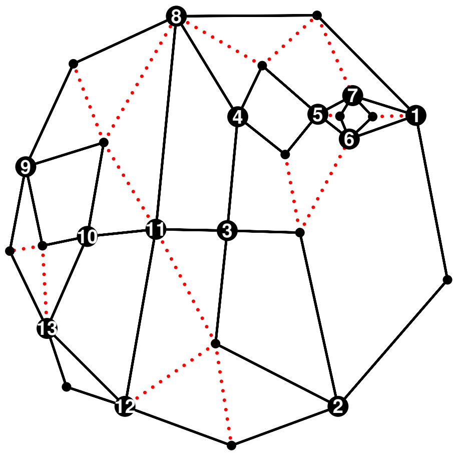
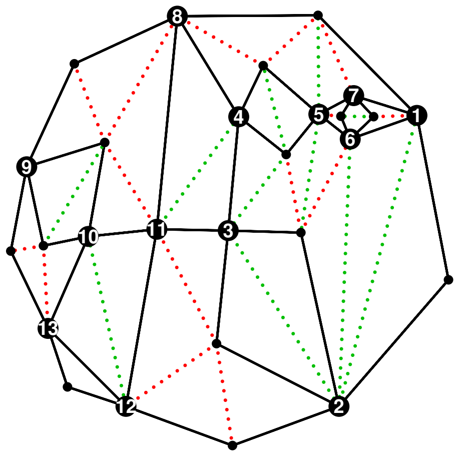


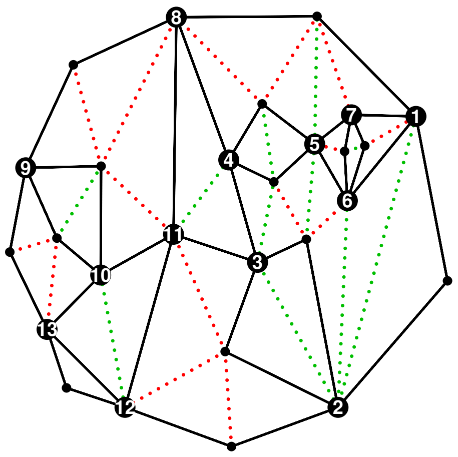
Figure 4: *Embedding 13ah_1131: (a) bounding polygon, (b) perimeter splitting, (c) first TE pass, (d) first augmentation step, (e) second TE pass, and (f) second augmentation step.*



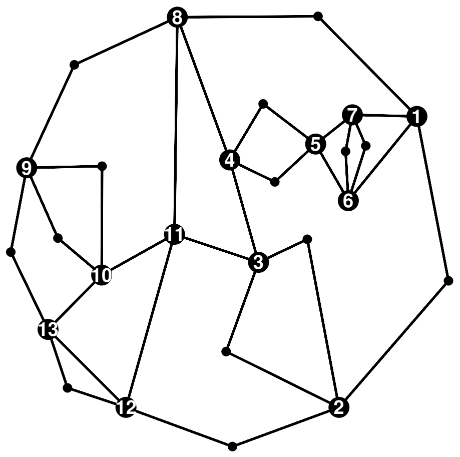
(g)



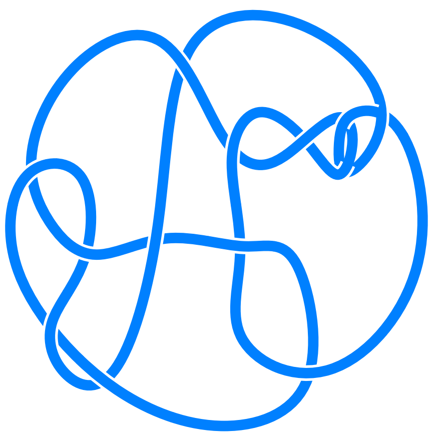
(h)



(i)



(j)



(k)

Figure 5: *Embedding 13ah_1131 (continued): (g) third TE pass, (h) triangulation, (i) final TE pass, (j) virtual edges removed, and (k) drawn without smoothing.*

Comparison

An advantage of ITE over the base TE approach described in [2] is that separation sets do not need to be calculated up-front, since they emerge as a by-product of each TE pass at negligible cost. This makes ITE quite efficient; the example shown above takes ≈0.8ms to compute on average, whereas the base algorithm takes almost twice as long at ≈1.5ms.[4] This discrepancy only increases with the number of crossings.

Another advantage of ITE is that geometric information grows incrementally as more vertices are successfully embedded with each TE iteration, allowing more informed decisions regarding which vertices to connect to during the graph augmentation steps. The embeddings produced by ITE tend to be at least as good as those produced by the base TE algorithm, and were actually better in all cases tested in terms of number of subsequent smoothing passes required to stabilise to a good final drawing.

The base TE algorithm [2] is more mathematically sound as it guarantees triconnectivity in relevant parts of the graph before TE is applied. The ITE approach does not guarantee triconnectivity until the process is complete and instead relies on a fortuitous side-effect of the algorithm with little theoretical justification. However, this side-effect works well in practice to allow good results more efficiently.

Conclusion

Iterated Tutte embedding (ITE) is a robust and efficient way to create planar embeddings for the underlying graphs of mathematical knots. It harnesses the power of Tutte embedding (TE) while exploiting a side-effect of TE to avoid the explicit calculation of separation sets up front. The resulting embeddings benefit from the additional geometric knowledge that is gained with each TE pass, and typically require minimal smoothing to produce aesthetically pleasing knot drawings. This approach is well-suited to interactive digital applications that must draw mathematical knots from their algebraic descriptions in real-time.

Acknowledgements

This work was supported by the European Research Council as part of the *Digital Ludeme Project* (ERC Consolidator Grant #771292) run by Cameron Browne at Maastricht University (http://ludeme.eu).

References

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

[2] C. Browne. "Drawing Knots with Tutte Embedding." *Bridges Proceedings*, Halifax, Canada, Jul. 27–31, 2023.

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

[4]Timings were done on a single thread of a standard consumer laptop machine with Apple M1 chip.