Visualizing the Computation Tree of the Tutte Polynomial

Bennett Thompson, David J. Pearce, and Craig Anslow*
Victoria University of Wellington, New Zealand

Gary Haggard†
Bucknell University, USA

Abstract

The computation trees of the Tutte Polynomial algorithm are very large. Understanding the effects of applying heuristics to the algorithm for example to classify *knots* is very challenging. We have constructed visualizations of the Tutte Polynomial computation tree. The visualizations are useful to study the effect of various heuristics on the algorithms' operation.

CR Categories: D.2.6 [Programming Environments]: Graphical Environments—; G.2.2 [Graph Theory]: Graph algorithms—[I.1.2]: Algorithms—Analysis of algorithms

Keywords: Software Visualization, Tutte Polynomial

1 Introduction

Tutte polynomials play an important role in graph theory, combinatorics, matroid theory, knot theory, and experimental physics. The Tutte Polynomial can be used to classify *knots*. A knot can be thought of as a tangled cord with the ends joined. If the tangled cord is a *trivial knot*, it could be untangled with the ends still fused; however, if the tangled cord is a *nontrivial knot*, a cut is needed to untangle it. The double helix which constitutes DNA can be visualized as two very long strands that are intertwined and coiled so much as to form a knot. In order for DNA to replicate it must first "untangle" itself and various enzymes are responsible for this. The type of knot involved affects this process, and a better understanding of this would yield additional insight into the replication, transcription, and recombination of DNA [Murasugi 1996].

Haggard *et al.*[Haggard et al. 2007] have developed the most efficient algorithm to compute the Tutte polynomial of a graph of sufficient size to represent a DNA knot [Bollobas and Riordan 1999]. The algorithm relies on various optimizations and heuristics to obtain good performance. However, the reason that a particular heuristic is effective often remains unclear. We have developed visualizations of the computation tree for the Tutte Polynomial to understand the heuristics and suggest better ones which will lead to a faster algorithm in practice.

2 Computing Tutte Polynomials

A *graph* is defined as a pair (V, E), where V is the *vertex set* and $E \subseteq V \times V$ the *edge set*. In this paper, we consider only *undirected* graphs, meaning (x, y) is the same as (y, x). A *loop* is an edge (x, x) between the same vertex, whilst a *bridge* is an edge whose removal disconnects two or more vertices (i.e. there is no longer a path between them). The *degree* of a vertex is the number of vertices incident on it.

The definition of a Tutte polynomial outlines a simple recursive procedure for computing it. We are free to apply its rules in whatever order we wish, and to choose any edge to operate on at each stage. Two operations are essential to understanding the Tutte polynomial definition. These are: edge deletion, $G - e$; and edge contraction, G/e. The latter involves first deleting e, and then merging its endpoints.

Definition 1. The *Tutte polynomial* of a graph $G = (V, E)$ is a two-variable polynomial defined as follows:

$$T(G) = \begin{cases} 1 & E(G) = \emptyset \quad (1) \\ xT(G/e) & e \in E \text{ and } e \text{ is a bridge} \quad (2) \\ yT(G-e) & e \in E \text{ and } e \text{ is a loop} \quad (3) \\ T(G-e) + & e \in E \text{ and } e \text{ is neither a} \quad (4) \\ \quad T(G/e) & \text{loop nor a bridge} \end{cases}$$

The order of applying the rules of Definition 1 significantly affects the size of the computation tree. An "efficient" order can reduce work in a number of ways. For example, there are two situations where an edge is associated with a factor directly: if the edge is a loop, the factor is y; likewise, if the edge is a bridge, the factor is x. Eliminating such edges as soon as possible and storing the factor (a cache of computed polynomials) for later incorporation into the answer reduces work by lowering the cost of operations (e.g. contracting, connectedness testing, etc.) on graphs in the subtrees below the removal.

The choice of edge for a delete/contract operation can also greatly affect the size of the computation tree. In particular, it affects the likelihood of reaching a subgraph isomorphic to one already seen. We have elsewhere developed two simple *edge selection heuristics* which appear to perform well. The first, called MINSDEG, minimises the degree of either end-point; that is, it chooses an edge where one endpoint has the smallest degree of any. The second, called VORDER, relies on an arbitrary ordering of the vertices; starting from the first vertex in the order, it continuously selects edges from the same vertex until none remain, before moving on to the next vertex in the ordering.

Understanding why the edge selection heuristics MINSDEG and VORDER perform so well is not easy. This is because the computation trees we are interested in typically have hundreds of thousands of nodes, and it is difficult to gauge exactly what effect each heuristic is having. Understanding them better would allow us to design better heuristics.

3 Visualizing the Computation Tree

We have developed a visualizer in Java for the Tutte Polynomial computation. We used the radial layout visualization technique for viewing the computation tree which is essentially a top down view of a cone and maximizes the use of screen real estate. We split the radial view into an arrangement of concentric circles divided into *wedges*. We call this the *wedge display*, as illustrated in Figure 1. The display provides an uncluttered view of the computation tree, with the flush proximity of nodes allowing node characteristics to be effectively summarized by colour.

Figure 2 illustrates two computation trees for the same starting graph, computed using the two different heuristics. It is immediately apparent from this visualization that the effect of the heuristics can be significant.

*E-mail: {djp,craig}@mcs.vuw.ac.nz
†E-mail: haggard@bucknell.edu

SOFTVIS 2008, Herrsching am Ammersee, Germany, September 16–17, 2008.
© 2008 ACM 978-1-60558-112-5/08/0009 $5.00

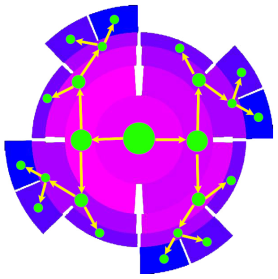

Figure 1: *Nodes in the computation tree are arranged into the wedge display where each node is represented by a* wedge *of colour, rather than a small circle, making better use of space.*

Figure 3 shows the visualizer's *application view*; this allows the user to manipulate their view of the computation tree through zooming, shifting, and other effects. The *target viewport* provides the main view window for the computation tree. Within this view the user can click and drag the view, zoom in/out and select a node, amongst other things. The *macro view* shows an outline view of the computation tree, which helps the user navigate the computation tree; a box in the macro view indicates the size and location of the target viewport. The user can also reposition the target viewport by clicking on the macro view directly. The *node view* shows the graph at a particular node in the computation tree, which is selected by clicking on the target viewport.

Figure 4 demonstrates another view produced by our visualizer that can provide some useful insight. This shows the distribution of matches in the cache. We have a cache that is used to store computed polynomials for intermediate graphs seen during the computation, so that they can be recalled when that intermediate graph is encountered again.

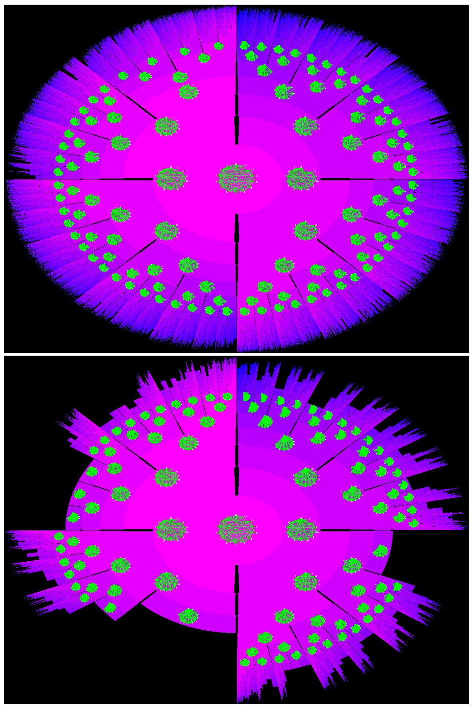

Figure 2: *Tutte polynomial computation of the same graph. The* VORDER *heuristic (bottom) produces a computation tree with 325K steps and the* MINSDEG *heuristic (top) produces 806K steps.*

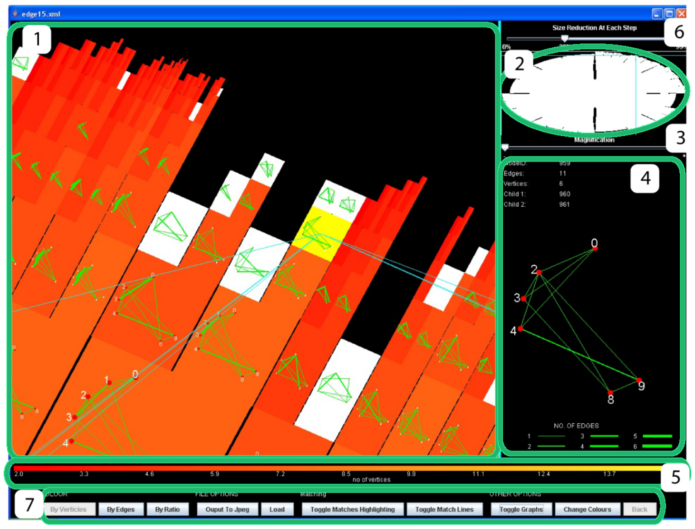

Figure 3: *The application view with the following labelled parts:* ***1.*** *Target View,* ***2.*** *Macro View,* ***3.*** *Magnification Slider,* ***4.*** *Node View,* ***5.*** *Colour Gradient Key,* ***6.*** *Squashing Slider,* ***7.*** *Function Panel.*

We have presented a visualization of the Tutte polynomial computation. This has proved useful in aiding understanding of the computation. We are unaware of any other work on visualizing the Tutte Polynomial computation. In the future, we would like to consider visualization of the computation tree for other NP-complete problems, such as SATisfiability (SAT) solving.

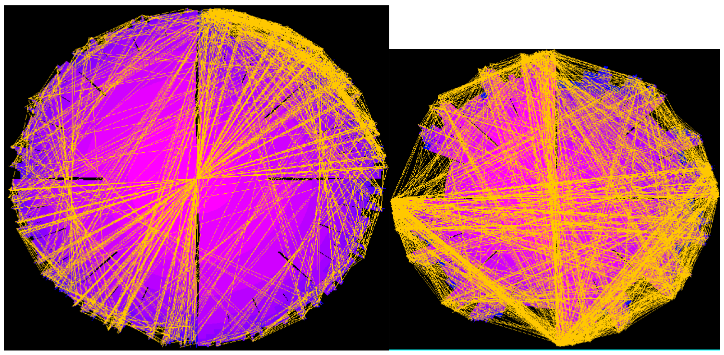

Figure 4: *Tutte Polynomial computation of the same graph (*MINSDEG *heuristic on the left and* VORDER *heuristic on the right). Each line shown on the two diagrams connects the point when an intermediate graph is first encountered and stored in the cache, with a later point where that graph is recalled and used.*

References

BOLLOBAS, AND RIORDAN. 1999. A tutte polynomial for coloured graphs. In *Combinatorics, Probability and Computing*, vol. 8. Cambridge University Press.

HAGGARD, G., PEARCE, D. J., AND ROYLE, G. 2007. Computing tutte polynomials. Tech. rep., Victoria University of Wellington.

MURASUGI, K. 1996. *Knot Theory and Its Applications*. Birkhauser.