A new edge selection heuristic for computing the Tutte polynomial.

Michael Monagan. Department of Mathematics, Simon Fraser University, British Columbia.

In [1] Garry Haggard and David Pearce computed the Tutte polynomial for the truncated icosahedron graph shown in the figure below right. It took their C++ code about one week to compute it on a grid of 150 computers. Using the edge selection and vertex ordering heuristics presented here we are able to compute it in less than 2 minutes in Maple on a single core of an Intel Core i7 desktop. The new heuristics appear to work well for all sparse graphs. But first, what is the Tutte polynomial and why is it of interest?

Definition (Tutte [2]). Let G be an undirected graph, possibly a multi-graph. Let e be any edge in G. Let $G - e$ denote the graph obtained by deleting e and let $G\,/\,e$ denote the graph obtained by contracting e, that is, first deleting e then joining e's vertices.

The Tutte polynomial, denoted $T(G, x, y)$, is defined by

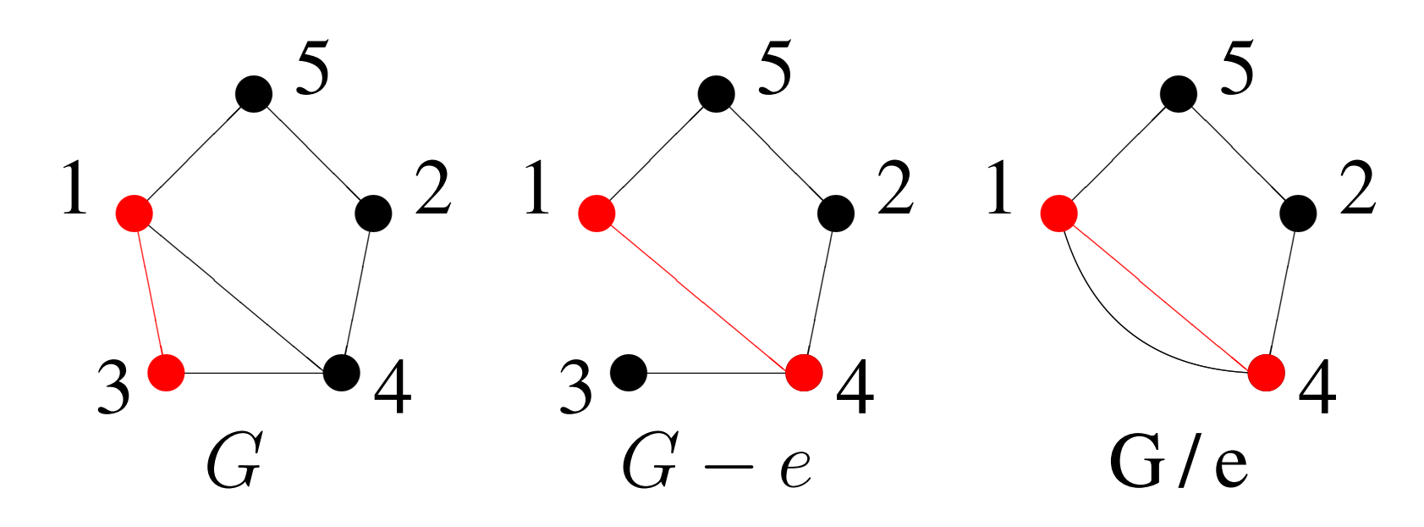
$$T(G) = \begin{cases} 1 & \text{if } G \text{ has no edges,} \\ x\,T(G\,/\,e) & \text{if } e \text{ is a cut-edge in } G, \\ y\,T(G - e) & \text{if } e \text{ is a loop in } G \\ T(G - e) + T(G\,/\,e) & \text{otherwise.} \end{cases}$$

It follows that $T(G, x, y)$ is a bivariate polynomial in x and y with integer coefficients. The coefficients measure connectivity. The Tutte polynomial is of interest because the chromatic, flow and reliability polynomials are special cases. It is known to be NP-hard to compute.

The definition gives a recursive algorithm for computing $T(G)$ known as the "edge-deletion-contraction" algorithm. The recursive calls in $T(G-e)+T(G\,/\,e)$ imply an exponential time complexity for computing it. If, however, we remember the Tutte polynomial for each recursive call in the computation tree, it may happen that we encounter a graph that we have already seen which could reduce the cost, possibly to polynomial time, for some families of graphs. In [3] Haggard, Pearce and Royle use the graph isomorphism test from Brendan Makay's nauty package to implement this idea. Roughly speaking, for random cubic graphs, this doubles the size of the graph they can handle in a given amount of time.

Which edge in G should we pick? Which choice will more likely generate graphs that we have seen before in the computation tree? In [3] Haggard, Pearce and Royle propose two heuristics called MAXDEG and VORDER. By trying variations on their VORDER heuristic we have found one that works much better. Moreover, it is sufficient to test for identical graphs in the computation tree only – so no graph isomorphism test is needed.

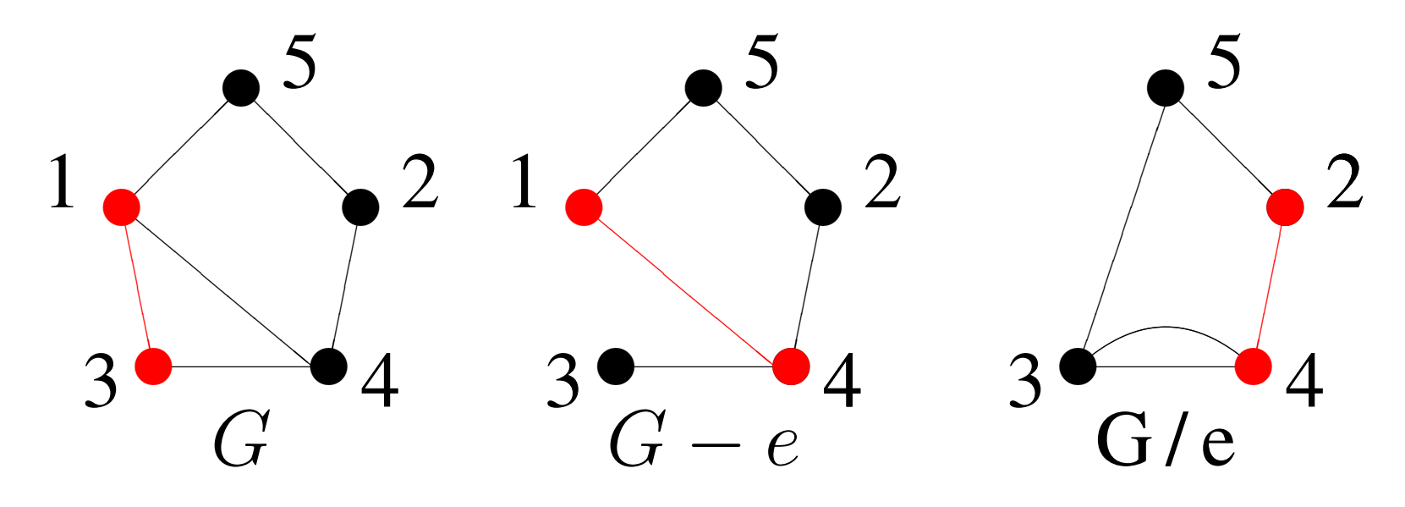
Two edge selection heuristics.



VORDER-pull

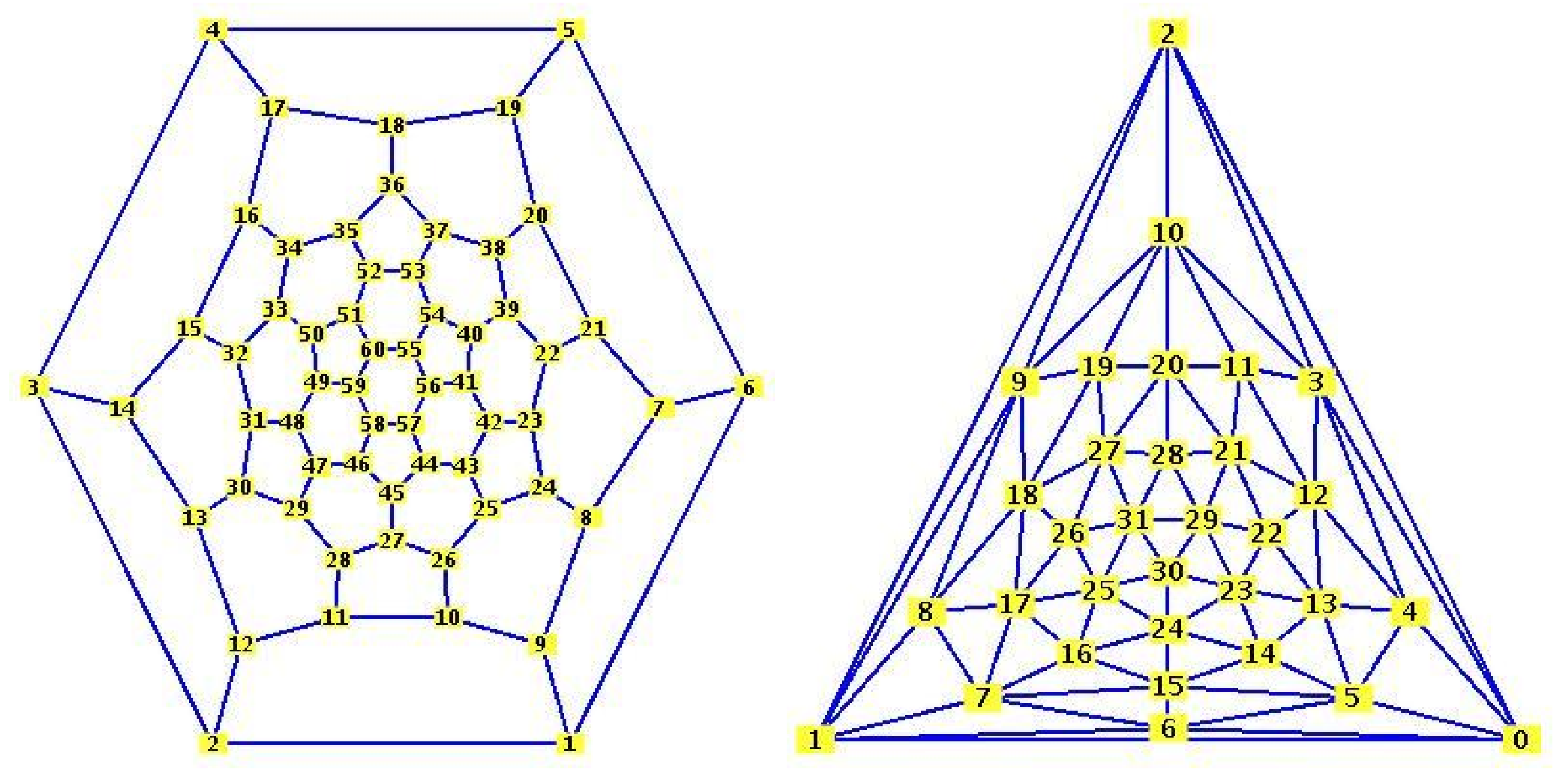
Consider the graph G shown in the figure above. The vertex order heuristic picks the edge $e = (u, v)$ where u is the first vertex in the G and v is the first vertex adjacent to u. In our example $u = 1$, $v = 3$, hence $e = (1, 3)$ is chosen. Shown are the graphs $G - e$ and $G\,/\,e$ where when we contracted the edge $e = (1, 3)$ we "pulled" vertex 3 down to vertex 1. The next edge selected in $G\,/\,e$ will be one of the edges (1,4).

There is alternative choice here when constructing $G\,/\,e$. Instead of "pulling" vertex $v = 3$ down to $u = 1$, if instead we "push" vertex $u = 1$ up to $v = 3$ we get the contracted graph shown in the figure below. Observe that the two contracted graphs $G\,/\,e$ in the figures are isomorphic. However, in the vertex order heuristic, the next edge selected in $G\,/\,e$ is different. Edge (2,4) is selected.



VORDER-push

The short-arc vertex order heuristic SHARC.



The truncated icosahedron graph G and its planar dual G^*.
Their Tutte polynomials are related by $T(G, x, y) = T(G^*, y, x)$.

$$T(G^*) = x^{31} + 59\,x^{30} + 60\,x^{29}y + 1710\,x^{29} + \ldots + 160271797870414\,y^2 + 11551226205884\,y$$

If you look at the vertex ordering in the truncated icosahedron graph above, you will see a cycle for vertices (1,2,3,4,5,6,1). The next three vertices (7,8,9) form a shortest path from the cycle back to the cycle, that visually looks like an arc. The next three vertices (10,11,12) form another shortest path from the set of vertices included so far back to itself. Repeating this gives an ordering on the vertices that we call a `short arc` ordering. Such an ordering can be computed in linear time using a breadth-first-search in G. See the paper for details.

What difference does all this make? It turns out it makes a huge difference. We find that VORDER-push is much better than VORDER-pull and the SHARC ordering is consistently better than a simple breadth-first-search ordering and much better than depth-first-search ordering. Why? The paper suggests one reason.

Timings and Maple implementation

We implemented the heuristics in Maple using a simple list of neighbors representation for G as shown in the figure below. Our software will become available in Maple's `GraphTheory` package (see [4]) for Maple 17.

We generated 10 random cubic graphs on n vertices and computed $T(G, x, y)$ using the MINDEG, VORDER-pull and VORDER-push heuristics. The first table is for a random vertex ordering. In the second table we relabeled the vertices using a SHARC ordering. Two timings, the median and average time, in CPU seconds, are reported. We used an Intel Core i7 desktop computer with 6 gigabytes of RAM. The data speaks for itself.

	MINDEG		VORDER pull		VORDER push	
n	ave	med	ave	med	ave	med
16	0.41	0.36	0.18	0.11	0.22	0.14
18	1.21	1.02	0.53	0.33	0.57	0.45
20	3.90	3.38	1.27	1.02	1.86	1.46
22	14.40	12.07	4.65	3.36	7.22	6.88
24	56.24	32.19	13.84	9.23	25.05	22.46
26	193.34	118.98	41.03	20.07	58.94	24.57
28			199.70	116.32	210.69	75.24

Timings (in seconds) for random cubic graphs with n vertices using random vertex order.

	MINDEG		VORDER pull		VORDER push	
n	ave	med	ave	med	ave	med
18	0.68	0.51	0.05	0.03	0.02	0.02
22	7.73	4.68	0.38	0.14	0.10	0.07
26	80.11	38.45	1.24	0.41	0.17	0.12
30			11.10	4.36	0.67	0.37
34			94.58	19.15	2.06	1.29
38					5.40	2.83
42					40.66	8.82
46					87.63	49.03
50					179.64	39.61

Timings (in seconds) for random cubic graphs with n vertices using SHARC vertex order.

References

[1] Gary Haggard, David Pearce, and Gordon Royle. Code for Computing Tutte Polynomials. `homepages.ecs.vuw.ac.nz/~djp/tutte`

[2] William Tutte. A contribution to the theory of chromatic polynomials. *Can. J. Math.* **6** (1954) 80−91.

[3] Gary Haggard, David Pearce, and Gordon Royle. Computing Tutte Polynomials. *Trans. on Math. Software* **37**:3 (2011) article 24.

[4] Jeff Farr, Mahdad Khatarinejad, Sara Khodadad, and Michael Monagan. A Graph Theory Package for Maple. *Proceedings of the 2005 Maple Conference*, pp. 260–271, 2005.