Using Thresholds to Compute Chromatic Polynomials

Gary Haggard
Bucknell University
Lewisburg, PA 17837

Thomas R. Mathies
Knox College
Galesburg, IL 61401

1 Abstract

An efficient algorithm for computing chromatic polynomials of graphs is presented. To make very large computations feasible, the algorithm combines the dynamic modification of a computation tree with a hash table to store information from isomorphically distinct graphs that occur during execution. The idea of a threshold facilitates identifying graphs that are isomorphic to previously processed graphs. The hash table together with thresholds allow a table look-up procedure to be used to terminate some branches of the computation tree. This table lookup process allows termination of a branch of the computation tree whenever the graph at a node is isomorphic to a graph that is stored in the hash table. The hashing process generates a large file of graphs that can be used to find any chromatically equivalent graphs that were generated. The initial members of a new family of chromatically equivalent graphs were discovered using this algorithm.

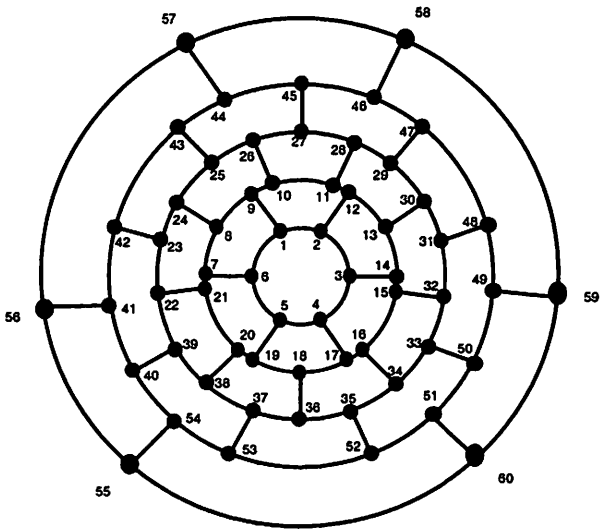
2 Introduction

A chromatic polynomial for a graph G is a polynomial that has as its value for each nonnegative integer n the number of ways G can be colored using n colors. The degree of a chromatic polynomial for a graph G is the number of vertices in G. In 1965 Hall, Siry, and Vanderslice [HSV] manually computed the chromatic polynomial of the planar dual of the truncated icosahedron. This is a graph with 32 vertices and 90 edges with each vertex of degree five or six. The lack of any computer implementation of an algorithm for this computation led the first author to work on this problem several times (unsuccessfully) since being introduced to the problem by Dick Wick Hall in 1969. Now the joint effort described here has led to a

ARS COMBINATORIA 58(2001), pp. 85-95

successful approach. The basic idea of the algorithm that was implemented starts with the well-known Delete–Contract Theorem. The algorithm and some of the subsequent enhancements that led to the computation of this chromatic polynomial are described in [Ha1], [HR], and [Ha3]. Following the successful computation of the chromatic polynomial of the planar dual of the truncated icosahedron, a new benchmark was needed to carry on this program of research. The graph shown in Figure 1 which is the cubic, planar representation of the truncated icosahedron filled the need. The chromatic polynomial of this graph is a degree 60 polynomial. The computation trees for this problem are an appropriate set to study. Future work will focus on how the algorithmic design ideas used to compute this chromatic polynomial can be applied to other computational problems.

Other approaches to developing algorithms for computing chromatic polynomials have been presented. In [Sh] the algorithm for finding chromatic polynomials uses chordal graphs for termination conditions. In [An] an algorithm is proposed using the Whitney expansion that is proven to compare favorably with the estimate in [Wi]. These and other implementations of algorithms for computing chromatic polynomials, such as [Re] and [NW], have provided experimental results for computations of only relatively small graphs. The work of [HVS] was too large for these implementations.



The Truncated Icosahedron–TI

Figure 1

The chromatic polynomial of TI can be found in [HM]. For purposes of experimentation and timing studies, we define the following subgraphs of TI:

$$TIn = < \{1, 2, ..., n\} > \quad \text{for } n = 1, 2, ..., 60.$$

3 The Algorithm

The problem is to compute the chromatic polynomial of a graph $G = (V, E)$ with vertex set V and edge set E. The major improvement to the basic delete-contract algorithm was a technique for breaking the problem into a set of problems associated with isomorphically distinct graphs of various sizes. A value between 1 and $|V|$ that determines the size of the graphs in a set of isomorphically distinct graphs is called a **threshold**. A set of thresholds is any subset of $1, 2, \ldots, |V|$ ordered from the largest to the smallest value chosen. Any value included in a set of thresholds is simply called a threshold. A threshold represents the maximum number of vertices allowed for any graph in a particular set of isomorphically distinct graphs. When a graph is reduced to a size less than or equal to the size of the threshold value that is being used, it is determined whether the graph is isomorphic to a previously found graph at this threshold. If the graph is not isomorphic to a graph previously encountered when this threshold was reached, the graph is further reduced until it reaches a size equal to a lower threshold value. When the smallest threshold value designated is reached, the chromatic polynomial of the remaining graph is computed using the algorithm described in [Ha2]. When the graph's chromatic polynomial is computed, this value is used in the computation of the chromatic polynomial of the graph from which it was formed. When a chromatic polynomial is computed for a graph at a particular threshold, the chromatic polynomial is stored in the hash table with the isomorphic invariant of the graph. This recursive process continues until the original graph has its chromatic polynomial computed. If a graph of a given threshold size is found to be isomorphic to a previously encountered graph, its chromatic polynomial is found in a table. This process of filtering the number of graphs passed on from one threshold value to a smaller threshold value can be viewed as terminating branches of the computation tree. The nodes of branches that contained representatives of the isomorphically distinct graphs for that threshold are then processed until they reach a smaller threshold. Thus the number of graphs to be passed to smaller threshold values is substantially reduced. The algorithm is given in Figure 2.

```
Enter edges of Graph and build Graph
Initialize arrays and threshold table. Allocate the first block of
    storage for the hash table.
Renumber Graph vertices using breadth first
    search (bfs) numbering
Delete – Contract(Graph, Threshold, CPolynomial)
Output CPolynomial /* chrom. poly. of Graph */

Delete – Contract(Graph, Threshold, CPolynomial)
  while(graph to process)
    If |V(Graph)| is less than or equal to Threshold
      Remove any vertex of degree 1 in Graph
      Remove any vertex of degree 2 in Graph in a triangle
      Find isomorphic invariant
      Check hash table for invariant
      If invariant is in the table
        Add previously calculated polynomial to CPolynomial
      else
        If (no smaller threshold)
          FindPoly(Graph, Polynomial) /* compute chrom. poly. */
        else
          /* NewThreshold largest threshold less than |V(Graph)|
          Determine NewThreshold
          Delete – Contract(Graph, NewThreshold, Polynomial)
        Add Polynomial with vertex deletions incorporated to CPolynomial
        Add invariant and Polynomial to hash table
        Get a Graph from the stack or set end of stack flag
    else
      Select vertex with largest bfs number, v
      Delete edge of (v, w) in Graph where w has
        the largest bfs numbering among all neighbors of v
      Copy Graph to Graph1
      Contract vertices v and w in Graph1
      Remove any vertex of degree 1 in Graph1
      Remove any vertex of degree 2 in Graph1 in a triangle
      Stack contracted Graph1
  end while
```

Reduction and Computation Algorithm

Figure 2

The isomorphism test used in the algorithm is the code of [Mc]. The procedure *FindPoly* calculates the chromatic polynomial of a graph passed to it. *FindPoly* acts on a connected graph and preserves this property as an invariant throughout the procedure. Therefore, it is necessary to know that *Delete – Contract* also preserves connectedness as an invariant. This needed invariance follows from the fact that the edge that is deleted or contracted is chosen in a particular way relative to a breadth first search numbering of the vertices of the graph.

FACT. Let *Graph* be a connected graph with a breadth first search bfs numbering. Let v be the vertex of *Graph* with largest bfs number and let $deg(v) > 1$. *Delete – Contract* leaves connectedness invariant after any delete or contract operation on the edge (v, w) where w is the vertex adjacent to v with largest bfs number.

PROOF. Let *Graph* be a connected graph with $|V(Graph)| = n$ vertices. Let bfs be a breadth first search numbering of its vertices. For any breadth first search bfs numbering $1, 2, \ldots, n$ of a connected graph, all the induced subgraphs

$$< \{1, 2, ..., k\} > \quad \text{for} \quad k = 1, 2, \ldots, n$$

are connected graphs. Let v be the vertex of *Graph* with the largest bfs number. Let w be the vertex adjacent to v with the largest bfs number. Since $deg(v) > 1$, removing the edge (v, w) does not disconnect the graph because $< \{1, 2, ..., v-1\} >$ is connected and v is adjacent to at least one of those vertices.

Since the contract graph is formed by first deleting the edge (v, w) and this does not disconnect the graph as shown above, it remains to show that identifying two vertices does not disconnect a connected graph. This follows since the contracted graph is just

$$< \{1, 2, \ldots, v-1\} > \cup \{(w, \alpha) | (v, \alpha) \in Graph\}$$

where $w \in \{1, 2, \ldots, v-1\}$.

For the case that the vertex v that has the largest bfs number in *Graph* has $deg(v) = 1$, the algorithm removes v and its incident edge leaving a connected graph. After computing the chromatic polynomial of the resulting graph, a factor of $\lambda - 1$ is introduced to complete the computation of the chromatic polynomial of the original graph.

4 Timing Studies

The idea of a threshold is the key to understanding how the algorithm operates. The algorithm does not say anything about how many thresholds to use or which ones to use. The empirical results show, however, that a good choice of even one threshold for a computation can substantially reduce the running time. This reduction in running time is seen to be more dramatic when larger sets of thresholds are used. The chromatic polynomial of TI was, in fact, not computed until the idea of thresholds was implemented. Using a set of thresholds, this chromatic polynomial can be computed in about two minutes. This section should give the reader a good idea of how thresholds are used in this algorithm and can perhaps be used in other algorithms. First we show how the placement of a single threshold value affects the running time of the algorithm. Second, we show how using a set of thresholds with more than a single element affects the running time. Finally, we show how the order of input also can affect the running time. Run times for the same graph can be very different depending on the computation tree used. In this algorithm the computation tree is determined by the order in which edges are input.

Table 1 gives an example of the difference in performance determined by choosing different possible values for a single threshold in computing $P(TI39, \lambda)$. The labels refer to the number of thresholds (Thresh.), the number of isomorphically distinct graphs (Iso.), and the number of graphs that are isomorphic to one of the isomorphically distinct graphs (Non Iso.).

Thresh.	Time Sec.	Iso. Classes	Non Iso. Matches	Thresh.	Time Sec.	Iso. Classes	Non Iso. Matches
39	30986	1	0	26	363	128	11020
38	30466	2	0	25	187	117	18496
37	17330	3	2	24	97	102	36884
36	15767	7	3	23	57	95	72556
35	16079	14	7	22	47	98	136393
34	12704	23	19	21	60	105	265363
33	9338	28	54	20	89	130	478437
32	7340	35	118	19	159	97	1077854
31	3360	33	299	18	280	116	2002520
30	2002	45	792	17	534	121	4204693
29	984	50	2011	16	993	134	8037352
28	755	83	3902	15	1548	126	13374507
27	587	123	6695	14	1548	126	13374507
All runs used the same computation tree.							

Running Time for Different Choices of One Threshold

Table 1

Table 2 shows the variability in running time and the number of non-isomorphic graphs generated (Iso.) when the algorithm uses more than one threshold (Thresh.). Using $TI39$ as an example, there are 26 possible thresholds between 39 and 14 inclusive of both extreme values. The rows of the table give performance data for several possible choices of thresholds.

$TI39$			
# Thresh.	Thresholds	Time	Iso.
0	39	30986	1
1	21	60	105
2	26 17	12	249
3	26 19 14	5	328
4	26 21 17 14	3	457
5	32 26 21 17 14	2	535
6	32 26 21 17 14 12	2	535
7	32 26 21 18 17 14 12	2	639
8	32 26 24 21 19 17 14 12	1	720
9	32 28 26 24 21 19 17 14 12	1	795
10	36 32 28 26 24 21 19 17 14 12	1	802
11	34 32 30 28 26 24 21 19 17 14 12	1	863
13	36 34 32 30 28 26 24 21 19 17 14 12 9	1	887
25	38 37 36 35 34 33 32 ... 18 17 16 15 14	1	1577
All runs used the same computation tree.			

Experimental Determination of Optimal Sized Threshold Set For $TI39$

Table 2

The results in Table 2 do not imply that the fourteen sets of thresholds contain the set of thresholds that give the shortest running time. The obvious question once the number of thresholds has been set is to ask which thresholds are most effective. In Table 2 the thresholds are just chosen to be distributed over all the candidates. In Table 3 there is a comparison among sets of thresholds with different edge orderings to generate different computation trees. The number of non-isomorphic graphs determines most of the running time since all other graphs have their chromatic polynomial found by a table looking procedure.

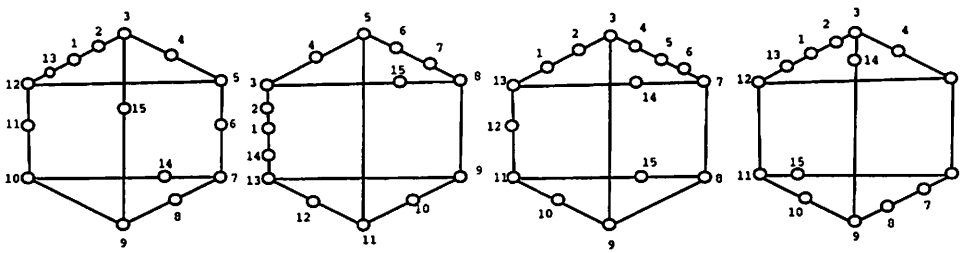
Thresholds	# non-isomorphic graphs at all thresh.	# copies
36 34 32 30 28 26 24 21 19 17 14 12	556	2555
	837	4144
	1264	6765
	1971	8710
36 34 32 30 26 24 22 20 18 16 14 12	577	2677
	786	3719
	819	3778
	4145	15714

Runs with Different Pernutations of the Edge Set

Table 3

5 Application

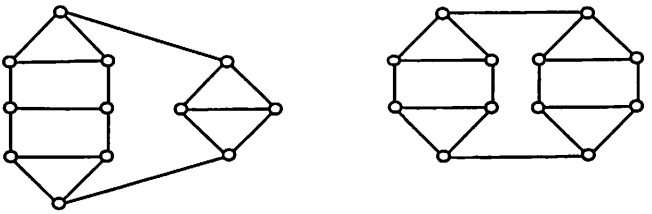
A problem of interest concerning chromatic polynomials is to determine whether or not two non-isomorphic graphs have the same chromatic polynomial. Such graphs are called **chromatically equivalent.** As a side effect of the algorithm presented, a file is generated that contains the isomorphic invariant and the chromatic polynomial of each isomorphically distinct graph encountered at each threshold level. Four chromatically equivalent graphs generated by an execution of the algorithm are shown in Figure 3.



Chromatically Equivalent Graphs with 15 Vertices

Figure 3

Two other chromatically equivalent graphs discovered when computing the chromatic polynomials of randomly generated cubic graphs are shown in Figure 4.



Chromatically Equivalent Ladders with 12 Vertices

Figure 4

With the information that the graphs in Figure 4 are chromatically equivalent, a study [GHR] has shown that these are just two examples of a whole family of graphs that is comprised of sets of chromatically equivalent graphs. For an even integer $2k$ where $k \geq 6$, there are $\lfloor k/2 \rfloor - 1$ chromatically equivalent graphs with $2k$ vertices in this family.

6 Summary

The complexity of the algorithm presented needs a worst-case analysis that is different from the work of either [Wi] or [Wa]. T For example, the analysis of [Wi] for the complexity of a delete-contract approach to computing chromatic polynomials has a resulting value the order of magnitude $O(1.62^{|V|+|E|})$ where V and E are the vertex and edge set of the graph considered. The assumption of this analysis is that the delete-contract process continue until $|V| + |E| \leq 1$. In addition, it is assumed that there are no efficiencies introduced for termination conditions such as stopping when a tree is generated as suggested in [NW] or even computation tree modifications as incorporated in the algorithm presented. It is not clear how to include modification to the computation tree into the analysis and so this remains an open question.

Using a combination of algorithm development and timing studies, an effective algorithm was developed for computing the chromatic polynomial of the truncated icosahedron. Computation trees are modified dynamically by deleting either edges or vertices of small degree [Ha3]. In addition, spanning trees in a graph are dynamically altered as a result of either vertex deletion or edge contraction. The focus on modifying a computation tree using thresholds shows the possibility of improving other algorithms provided there is an effective isomorphism for the structure involved. In addition the algorithm provided a number of new examples of chromatically equivalent graphs that led to the discovery of the existence of a family of chromatically equivalent graphs.

Bibliography

[An] Martin H.G. Anthony, Computing chromatic polynomials, *Ars Combin.* 29C (1990), 216-220.

[GHR] Stephen Guattery, Gary Haggard and Ronald C. Read, Chromatically Equivalent Cycles of Ladders, private communication.

[Ha1] Gary Haggard, Computing Chromatic Polynomials of Large Graphs, *Congressus Numerantium* 119, 1996, pp.113-122.

[Ha2] Gary Haggard, Computing Chromatic Polynomials of Large Graphs I, *The Journal of Combinatorial Mathematics and Combinatorial Computing*, 1993(13), pp.175-186.

[Ha3] Gary Haggard, Spanning Trees and Vertex Deletion, *Congressus Numerantium* 112 (1995), pp. 33-36.

[HM] Gary Haggard and Thomas R. Mathies, The computation of chromatic polynomials, *Discrete Mathematics* 199 (1999), pp. 227-231.

[HR] Gary Haggard and R.C. Read, Isomorphism Abstract Data Type for Small Graphs, *Caribbean Journal of Mathematics and Computer Science* 3 (1 & 2), (1993), 35-43.

[HSV] D. W. Hall, J. W. Siry, and B. R. Vanderslice, The Chromatic Polynomial of the Truncated Icosahedron, *Proceedings of the American Mathematical Society* (1965), 620-628.

[Mc] B. D. McKay, nauty User's Guide (Version 1.5), *Technical Report TR-CS-90-02*, Australian National University, Department of Computer Science, 1990.

[NW] A. Nijenhuis and H.S. Wilf, *Combinatorial Algorithms*, Academic Press, New York (1975).

[Re] Ronald Read, An improved method for computing the chromatic polynomials of sparse graphs, *Research Report CORR 87-20, Dept. of Comb. and Optim.*, U. of Waterloo, 1987.

[Sh] D. R. Shier and N. Chandrasekharan, Algorithms for computing chromatic polynomials, *J. Combin. Math. Combin. Comput.* 4 (1988), 213-222.

[Wa] Timothy R. Walsh, Worst-Case Analysis of Read's Chromatic Polynomial Algorithm, *Ars Combinatoria* 46 (1997), pp. 145-151.

[Wi] Herbert S. Wilf, *Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ (1986).