Computation of chromatic polynomials using triangulations and clique trees

P. Berthomé[1], S. Lebresne[2], and K. Nguyễn[1]

[1] Laboratoire de Recherche en Informatique (LRI), CNRS UMR 8623, Université Paris-Sud, 91405, Orsay-Cedex, France, {Pascal.Berthome,Kim.Nguyen}@lri.fr
[2] Preuves, Programmes et Systèmes (PPS), CNRS UMR 7126, Université Paris 7, Case 7014, 2 Place Jussieu, 75251 PARIS Cedex 05, France, Sylvain.Lebresne@pps.jussieu.fr

Abstract In this paper, we present a new algorithm for computing the chromatic polynomial of a general graph G. Our method is based on the addition of edges and contraction of non-edges of G, the base case of the recursion being chordal graphs. The set of edges to be considered is taken from a triangulation of G. To achieve our goal, we use the properties of triangulations and clique-trees with respect to the previous operations, and guide our algorithm to efficiently divide the original problem. Furthermore, we give some lower bounds of the general complexity of our method, and provide experimental results for several families of graphs. Finally, we exhibit an original measure of a triangulation of a graph.
Keywords: Chromatic polynomial, chordal graphs, minimal triangulation, clique tree.

Résumé Dans cet article, nous présentons un nouvel algorithme pour calculer le polynôme chromatique d'un graphe quelconque G. Notre méthode s'appuie sur l'ajout récursif d'arêtes et de contraction de non-arêtes de G, la récursion s'arrêtant pour les graphes triangulés. L'ensemble des arêtes que l'on considère dans ce processus forme une une triangulation de G. Afin de calculer efficacement ce polynôme, nous étudions l'impact de l'ajout d'arêtes et de la contraction sur un arbre de cliques. Notre algorithme est alors guidé par ces opérations afin de diviser le problème le plus rapidement possible.

De plus, nous donnons quelques bornes inférieures induites par notre méthode. Nous fournissons par ailleurs une étude expérimentale de notre algorithme sur plusieurs familles de graphes.

Enfin, nous mettons en évidence un nouveau paramètre concernant les triangulations de graphe.
Mots-Clés : polynôme chromatique, graphes triangulés, triangulations minimales, arbres de cliques

1 Introduction

Introduced by Birkhoff and Lewis in 1946 [4], the chromatic polynomial of a graph G counts the number of ways of properly coloring G. This polynomial also captures many combinatorial information about a graph, describing acyclic orientations, the all-terminal reliability, and the spanning trees. More surprisingly, this polynomial is closely related in physics with the zero-temperature partition function of the q-state Potts antiferromagnet, motivating the computation this polynomial for some class of graphs by physicists (see, for example [6]). Created in order to give some proof of the famous 4-color theorem, the chromatic polynomial has been studied for itself. Studies include among others the search of the real/complex roots of the polynomial [14, 6], and the search of graphs uniquely defined by their chromatic polynomials [7].

Once we have the chromatic polynomial of any graph, its chromatic number, i.e., the minimum number of colors needed to properly coloring the graph, is simply the first integral non-zero of the polynomial, thus, this can be computed in polynomial time. However, since the general determination of the chromatic number of a graph is NP-complete [10, GT4], the determination of the chromatic polynomial induces heuristic (or exponential) methods, unless P=NP. As shown in [17], the chromatic polynomial includes many other notions than the chromatic number, thus its computation reveals to be quite complex even when the chromatic number is known, e.g., in the bipartite graphs. Conversely, it is clear that the class of graphs for which the chromatic polynomial is easy to determine is included in the class of graphs for which the determination of the chromatic number is easy. Recently, it was shown that computing the coefficients of this polynomial in general graphs is #P-hard [16].

Several papers deal with the effective computation of this polynomial. Many of them use the paradigm of edge contraction/deletion in order to develop an exponential time algorithm. The main strategies developed in the literature stop the recursion to graphs for which the polynomial is easy to compute or is already known. We can note the work of Haggard [13, 11, 12], in which this latter paradigm is used at the extreme: during the computation tables of chromatic polynomials are made and, at each step, if the considered graph (a minor of the original graph) is small enough, the method first checks whether the chromatic polynomial of an isomorphic graph has already been computed during the beginning of the process. This method allowed the author to compute the chromatic polynomial of a planar graph having 60 vertices, the so-called *truncated icosahedron*. This method is very efficient when the number of isomorphic minors is very important. Other papers, such as [17], use as base case chordal graphs for which the chromatic polynomial is easy to determine [5].

In this paper, we also use the chordal graphs as base cases in the recursion tree. However, we improve the global computation by exploiting the clique-tree associated to the considered chordal graphs. Another important difference with most of the literature is that we prefer an edge addition/contraction method rather than the classical edge contraction/deletion method.

In the remaining of the paper, we recall in Section 2 known results on the chromatic polynomials, chordal graphs and the triangulation of a graph. In Section 3, we provide results that lead to our algorithm for computing the chromatic polynomial in Section 4 and give lower bounds of the time complexity in Section 5. In Section 6, we provide some experimental results and comparison with other methods. Finally, we conclude the paper in Section 7. In Appendix A, we provide a complete proof of Theorem 1.

2 Preliminaries

In the following, we use classical graph theory on undirected graphs. Notions may be found for example in [2]. The notation $G_1 \sim G_2$ will denote that both graphs are isomorphic; the notation $\overline{G}$ denote the complement of $G = (V, E)$, i.e., $\overline{G} = (V, E')$, with $E' = \{(x, y) \notin E\}$.

2.1 Chromatic polynomials

Let $G = (V, E)$ be an undirected loop-less graph, with $n = |V|$ and $m = |E|$. A proper coloring of G with k colors is simply a function ϕ from V into $I_k = \{1, \ldots, k\}$ ($I_0 = \emptyset$) such that two neighbors in the graph have different colors. For a given integer $\lambda \geq 0$, the *chromatic polynomial*[1] $P(G, \lambda)$ is the number of distinct proper colorings of G using at most λ colors. In [17], the different forms of this polynomial are reviewed. According to the chosen basis, the coefficients reflect different properties of the graph. As example, let give some well-known chromatic polynomials, leading to the most popular basis for writing these polynomials.

Name		**Chromatic polynomial**
Empty graph	$\bar{K}_n$	λ^n
Complete graph	K_n	$\lambda^{(n)} = \prod_{i=0}^{n-1} (\lambda - i)$
Trees	T_n	$\lambda(\lambda - 1)^{n-1}$

Here are some simple results that are the base of the computation of the chromatic polynomial. If e is an edge of G, we note $G - e$ the graph obtained by removing e from E, and G/e, the graph obtained by contracting e. If e is not an edge of G, we denote $G + e$ the graph obtained by adding e to G. In this case, we can define G/e by $(G + e)/e$. Using this notation, the two following relations can be easily established.

$$P(G, \lambda) = P(G + e, \lambda) + P(G/e, \lambda), \text{ if } e \notin E \tag{1}$$

$$P(G, \lambda) = P(G - e, \lambda) - P(G/e, \lambda), \text{ if } e \in E \tag{2}$$

[1] It is sometimes called *chromial* as in [8].

Both formulations can be applied to compute recursively the chromatic polynomial. Base case in Formulation 1 is the complete graph, since we add edges to the initial graph, and to any contracted graph, whereas the base cases with the other formulation are the trees or the empty graphs. If the computations are using only one formulation and only one base case, we can derive directly the coefficients of the polynomials using the corresponding base. However, these formulations clearly lead to an exponential exploration and is not practical in many simple cases. For example, even for simple graphs as rings, the number of recursion steps, i.e., the size of the recursion tree is $O(2^{n^2})$ as explained in Section 5[2].

Note that these two equations simply verify that the object we are talking about is a polynomial, since it is obtained by a finite combination (addition/subtraction) of elementary polynomials. Algorithm 1 presents formally this algorithm.

Algorithm 1: **Primitive-Chromatic(G)**

```
    ▷ $G = (V, E)$ be a graph
    ▷ Returns the Chromatic polynomial of $G$
1   begin
2       if $G$ is a complete graph $K_n$ then Return $\lambda^{(n)}$
3       Let $e \notin E$, $G_1 = G + e$ and $G_2 = G/e$.
4           $P_1 \leftarrow$ Chromatic-Polynomial($G_1$)
5           $P_2 \leftarrow$ Chromatic-Polynomial($G_2$)
6           return $P_1 + P_2$
7   end
```

In order to improve the computation of this polynomial, we split the problem into smaller problems using the following simple folklore lemmas.

Lemma 1. *Let G be a graph composed by two connected components G_1 and G_2, then we have:*

$$P(G, \lambda) = P(G_1, \lambda)P(G_2, \lambda) \tag{3}$$

Using this lemma, we can find again the chromatic polynomial of the empty graph. Another simple consequence of this lemma is that the most interesting study only concerns connected graphs.

Lemma 2 ([14]). *Let G be a graph and G_1 and G_2 be subgraphs of G such that $G = G_1 \cup G_2$ and $G_1 \cap G_2 \sim K_r$, then we have:*

$$P(G, \lambda) = \frac{P(G_1, \lambda)P(G_2, \lambda)}{\lambda^{(r)}} \tag{4}$$

Using this lemma, we can derive the chromatic polynomial of the trees, by eliminating all the leaves, one by one. Another application is the computation of the chromatic polynomial of the chordal graphs as shown below. Lemma 1 may be

[2] Actually, this number is related to the number of partitions of an n-set into blocks of size > 1 [18, Seq A000296].

seen as special case of Lemma 2 by assuming that the chromatic polynomial of the null graph (no vertex) is 1.

In this paper, we will use this lemma to break down the effective complexity of computation of the chromatic polynomial. The goal in the recursion steps of our algorithm is first to complete separators of the graph in order to split the computation of the chromatic polynomial of a large graph into several computations for smaller ones. The edges that should be added in this process should belong to some triangulation of the input graph.

2.2 Triangulation and clique trees

A *chordal graph* G is a graph in which there are no induced cycles of length > 3. This class of graphs is very large and includes many subclasses such as trees, complete graphs, interval graphs. Furthermore, many problems that are NP-complete in general graphs are polynomially solvable in the chordal graphs, such as the colorability problem [10, GT4]. One useful representation of a chordal graphs is the *clique-tree*. The following definition is derived from [1].

Definition 1. *A clique-tree of a given chordal graph* $G = (V, E)$ *is a tree* $T = (\mathcal{V}, \mathcal{E})$ *such that:*

- $\mathcal{V} = \{C_1, C_2, \ldots, C_j\}$ *is the set of the maximal cliques of* G*;*
- *for every vertex* v *in* G*, the set of maximal cliques containing* v *induces a subtree of* T.

A simple example is given in Figure 1. It has been shown that there may exists several non-isomorphic clique trees of the same chordal graph [9]. Note that any edge in the clique tree represents a minimal separator of G. This provides a simple method to compute the chromatic polynomial for this family of graphs as shown in Lemma 3 below.

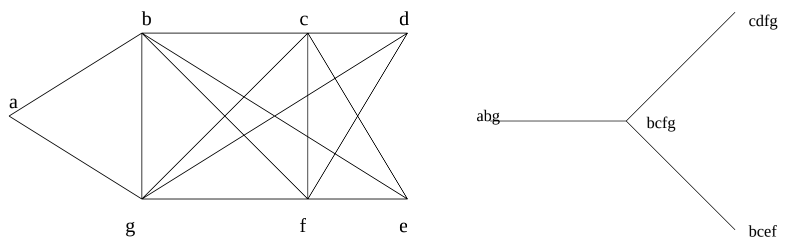


Figure 1. A chordal graph and its clique tree

For a general graph $G = (V, E)$, a *triangulation* of G is a set of edges F such that $G' = (V, E \cup F)$ is chordal. Finding a triangulation with the minimum number of edges is known as the *minimal fill-in* problem and is NP-complete [19]. A triangulation is minimal if no proper subset of it is a triangulation. Many efficient algorithms have been designed for computing a minimal triangulation. For example, the minimum degree heuristic and its variations [3] appear to be very efficient in practice for the minimum fill-in problem, however it does not constitute an approximation algorithm. In [15], a $O(k)$ approximation polynomial algorithm is given, where k is the minimum fill-in value.

In this paper, we define another measure of a triangulation, that will have a direct connection with the computational complexity of our method. Before this, we need to introduce the *augmented clique tree*, an useful data structure for our computation process.

Definition 2. *Let $G = (V, E)$ be a graph and F a minimal triangulation of G. Let $T = (\mathcal{V}, \mathcal{E})$ a clique tree of $G' = (V, E \cup F)$. The **augmented clique tree** of G for F is $T = (\mathcal{V}, \mathcal{E}, \phi)$, where ϕ is a labeling of $\mathcal{E}$ by subsets of F defined by:*

$$\phi(V_i, V_j) = F \cap E(G'[V_i \cap V_j]),$$

where $G'[U]$ denotes the subgraph of G' induced by U a subset of $V(G')$.

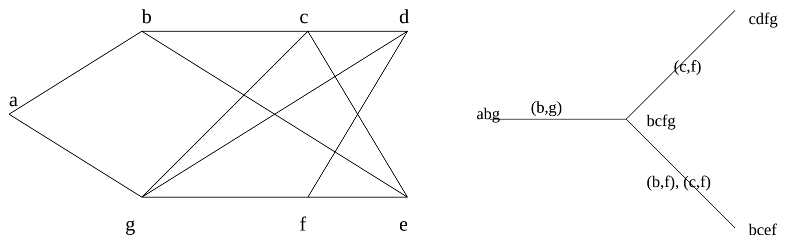


Figure 2. A graph and its augmented clique tree for the triangulation $F = \{(b, f), (b, g), (c, f)\}$

Note that the edges of T constitute the minimal separators of G', the triangulation of G. Consequently, the labeling defines for any edge of T the subset of F that has to be added in G to complete this separator.

Definition 3. *Given a graph G and a minimal triangulation F of G, we define the **thickness of the triangulation** F as:*

$$Th(G, F) = \max_{(V_i, Vj) \in \mathcal{E}} |\phi(V_i, Vj)| \tag{5}$$

*As extension, we define the **thickness of triangulation of the graph** G as the minimum for all the triangulations of G of the thickness of triangulations.*

$$Th(G) = \min_{F \text{ a triangulation of } G} Th(G, F) \tag{6}$$

For example, in Figure 2, the thickness of the proposed triangulation is 2. We will show in Section 7 that the thickness of a graph is an original parameter of the triangulation of a graph and that it is disconnected to the minimum fill-in.

Lemma 3 ([5]). *Let $G = (V, E)$ be a chordal graph. Let $T = (\mathcal{V}, \mathcal{E})$ be a clique-tree representation of G, $\mathcal{V} = \{V_1, \ldots, V_k\}$. Then, the chromatic polynomial of G is:*

$$P(G, \lambda) = \frac{\prod_{i=1}^{k} \lambda^{(|V_i|)}}{\prod_{(V_i,V_j)\in\mathcal{E}} \lambda^{(|V_i \cap V_j|)}} \tag{7}$$

Proof. This is a direct consequence of Lemma 2 applied recursively to any leaf of the clique tree.

□

Note that the form of the chromatic polynomial of any chordal graph is of the form:

$$P(G, \lambda) = \prod_{i=0}^{k} (\lambda - i)^{\alpha_i} \tag{8}$$

where the α_i's are non-negative integers. The converse is not true as shown in [7], where a family of non-chordal graphs has been exhibited with the same type of chromatic polynomials as in Equation 8. However, the coefficients α_i all equal to 1 define the complete graphs and ($\alpha_0 = 1, \alpha_1 = n - 1, \alpha_i = 0$ define the trees. One interesting question should be to determine the vectors α verifying the property of uniquely defining a family of chordal graph.

3 Evolution of the clique tree

In this section, we exhibit some properties of the augmented clique tree under the edge contraction and graph separation operations.

3.1 Clique tree and edge contraction

Let's consider a computation tree of the chromatic polynomial based on Equation 1 as shown in Figure 3. The root node will compute the global polynomial of G, its left child represent $G + e$ for a given non edge e of G and the right child G/e, for the same e. Then, this scheme appears until the node has to compute a base case. In this latter case, the node is a leaf of the computation tree and the corresponding graph is a base case, i.e., a complete graph if no optimization is provided or a chordal graph as explained in the following.

Since a complete graph is also a chordal graph, we can prune this computation tree such that all the leaves are chordal graphs. The main goal of our algorithm is to find a way to pre-compute this computation tree in such a way that it becomes

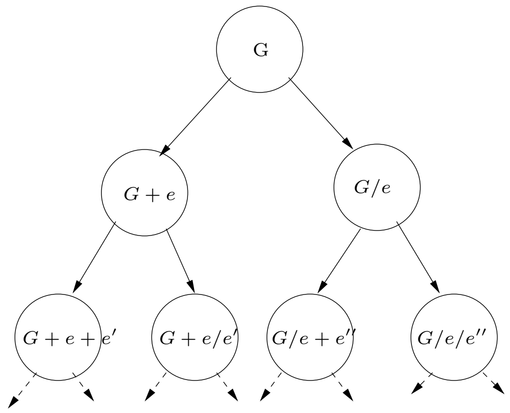


Figure 3. Computation tree for the chromatic polynomial

minimal. One efficient way to do that is to compute a single triangulation of G, say F, and operate on this set in the same time as operating on the graph. At each node of the computation tree, we associate three elements:

1. the graph G_1 whose chromatic polynomial has to be computed;
2. a triangulation of G_1, say F_1, and
3. the clique tree T_1 of $G'_1 = G_1 + F = (V(G_1), E(G_1) \cup F_1)$, i.e., an augmented clique tree of G_1 and F_1.

Then, we obtain the following theorem.

Theorem 1. *Let G be a graph, F a triangulation of G, and T a clique tree of $G+F$. Let e be an element of F. Let $G_1 = G+e$ and $G_2 = G/e$. Then, there exists F_1, F_2, T_1 and T_2 such that:*

1. *F_1 is a triangulation of G_1 and T_1 is a clique tree of $G_1 + F_1$;*
2. *F_2 is a triangulation of G_2, and T_2 is a clique tree of $G_2 + F_2$.*

All these sets can be computed in quadratic time from G, F and T.

Proof. This proof is divided into two parts.

The **first** one considers the graph $G_1 = G + e$. From the previous remarks, it is easy to see that $F_1 = F - e$ and $T_1 = T$ is a valid choice. These two sets can be clearly obtained in linear time.

The **second** part of the proof concerns $G_2 = G/e$. We will just sketch the proof here, for the sake of clarity. A detailed proof for this case is more tedious and is given in Appendix A. First, let us remark that if a graph is chordal, then it remains chordal after an edge contraction. In our case, $G+F$ is chordal, and consequently, so is $(G+F)/e$. Let denote a and b the extremities of e. The operation, $/e$ on a set or a graph simply consists in identifying both extremities of e and remove redundant elements.

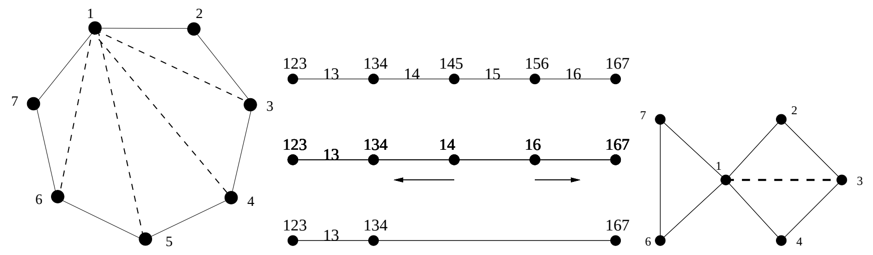
Then F_2 can be defined as $E((G+F)/e) - E(G/e)$. It simply corresponds to the elements of F/e that are edges of G/e. This occurs when an edge of type (b,x) in F collapses into an edge (a,x) in $E(G)$. Then, the clique tree corresponding to $G_2 + F_2$ can be obtained from the description of this graph. However, there exists a clever way to do this, simply based on the description of T. This method is described in Algorithm 2. This algorithm and the contraction algorithms can be performed in $O(n^2)$ operations, where n is the number of vertices of G.

□

Algorithm 2: **Clique-Tree-Contraction(T,e)**

```
    ▷ $T$ is a clique-tree of a given chordal graph $G$ and $e = (i,j)$ is an edge of $G$
1   begin
2       Within any clique set, replace the elements $j$ by $i$
3       while there exists two neighbor cliques $c_1$ and $c_2$ such that $c_1 \subseteq c_2$
4           Contract the edge $c_1, c_2$ in $T$
5       return $T'$, the resulting tree after renaming and contractions
6   end
```

We illustrate our method on a simple example in Figure 4. Let consider the cycle with 7 vertices and one of its minimal triangulation in Figure 4(a). Figure 4(c) shows the result on C_7 of the contraction of the edge $(1,5)$. Figure 4(b) shows how the (augmented) clique tree evolves using Algorithm 2.



(a) The cycle C_7 and one triangulation F

(b) Evolution of a clique tree

(c) $C_7/(1,5)$

Figure 4. The clique-tree contraction on the cycle. The triangulation sets are shown in dashed lines.

3.2 Clique tree and graph separation

We have previously seen that we can stop the computation once we reach a chordal graph, thus yielding a complexity which is directly related to the minimum fill-in of the graph. But this technique leads to another optimization. Let's consider the computation tree of the general computation of the chromatic polynomial shown in Figure 3 in which the first objective is to complete an edge of the augmented clique tree. This leads to a computation tree in which only the edges that fill a minimum separator of the triangulation of the graph are concerned. For any leaf of this computation tree, we obtain some graph for which a given separator is filled, at least the separator concerned by this "local" operation. Others may have been also affected since some edges of the triangulation may appear in several separators in the clique tree, as shown by Theorem 1. As consequence we can use Lemma 2 to divide the problem of the computation of the chromatic polynomial of this "leaf" graph into the computation of two smaller graphs for which the clique tree is smaller as shown in the following theorem.

Theorem 2. *Let G be a graph and G_1 and G_2 be subgraphs of G such that $G = G_1 \cup G_2$ and $G_1 \cap G_2 \sim K_r$. Let F be a triangulation of G and T be the clique tree of $G + F$. Then, the triangulation set F can be divided into two disjoint sets F_1 and F_2 such that F_1 (resp. F_2) is a triangulation of G_1 (resp. G_2). Furthermore, the clique trees of the triangulated graphs can be obtained from T by removing only one edge.*

Proof. First, since $G_1 \cap G_2 \sim K_r$, there is no edge of F that can be added to $G_1 \cap G_2$. Consequently any edge of F has to be added either on $G \backslash G_2$ or $G \backslash G_1$. Then let consider F_1 (resp. F_2) the subset of F for which both extremities are in G_1 (resp.F_2). Using the above remark, these two sets form a partition of F.

Considering a clique tree of $G + F$, since $G_1 \cap G_2$ is isomorphic to K_r, it corresponds to a separator of G, consequently for $G + F$. Thus, there exists an edge e in this tree that represents this separator. Consequently, removing this edge from the tree creates two clique trees T_1 and T_2. It is straightforward to see that T_i is a triangulation tree of $G_i + F_i$, $i = 1, 2$.

□

These two operations performed on an initial triangulation of the input graph lead to our general algorithm presented in the following section.

4 General algorithm

Using the results of the previous section, we present an algorithm for computing the chromatic polynomial of a given graph G as well as some bounds on its complexity. The idea of the algorithm is quite simple. First, using Theorem 1 we pre-compute a triangulation for a given graph G and use it to add (and contract) edges, the computation holds until a chordal graph is reached. The second optimization is to direct the choice of the edge in the addition/contraction algorithm in order to arise to a separator. For this choice, we use the augmented clique tree. Actually, if a label of the augmented clique tree is empty, the corresponding edge is a separator of the graph associated. The choice of the edge to add/contract at each step has a direct impact on the efficiency of the algorithm. We discuss the impact of the choice function in Section 4.2.

4.1 Algorithm

Algorithm 3: Chromatic-Polynomial(G, T)

```
▷ T is an augmented clique-tree of the graph G
▷ Returns the Chromatic polynomial of G
begin
    if $G$ is triangulated then return ChromaticPolynomial(G,T) using Lemma 3
    if $\exists e \in T$ such that $\phi(e) = \emptyset$
        Decompose $G$ using Theorem 2:
            Let $G_1$, $T_1$, $G_2$, $T_2$ and $K_r$ the resulting elements
            $P_1 \leftarrow$ Chromatic-Polynomial($G_1$, $T_1$)
            $P_2 \leftarrow$ Chromatic-Polynomial($G_2$, $T_2$)
            return $P_1 \times P_2 / P(K_r)$
    Let $e$ = ChoiceFunction($G$,$T$)
        Using Theorem 1, we compute
        $G_1 = G + e$, and the resulting $T_1 = T$
        $G_2 = G/e$, and the resulting $T_2$
        return Chromatic-Polynomial($G_1$, $T_1$) + Chromatic-Polynomial($G_2$, $T_2$)
end
```

The correctness of this algorithm is straightforward given Equation 1 and Theorems 1 and 2.

4.2 The choice function

The choice function used in Step 9 has a great impact on the overall efficiency of the algorithm. Let us remind that the main idea of our method is to add all the edges of $\phi(e)$ for a given edge e of the augmented clique tree, in order to create a clique on the graph and then, be able to split it into two smaller graphs. Nevertheless, to have this method perform efficiently, we need to separate the graph so that the number of edges added to each graph is evenly distributed.

For example, we can clearly see that if one choses always to add edges (of the set labeling an edge) at the extremity of the augmented clique tree, this will lead to a computation of exactly the same complexity as the one of Algorithm 1.

The function we use in the implementation chooses the edge e of the augmented clique such that the cardinal of the union of labellings of the sub-tree in "the right" and in "the left" of e are the closest (we try to be as fair as possible *a priori*). And for this particular edge e, we add all the edge of $\phi(e)$ and then separate the graph. We can remark that it is not necessarily the best choice, for at least two reasons. The first one is that we don't consider the cardinality of each set labeling the edge of the augmented clique tree. Secondly, merges can occur during the edges contraction and can change these sets, making them still correct but less optimal.

Thus, taking these two points into account when implementing the choice function, we can clearly improve our algorithm. However, it's unlikely that there is a best choice function for all graphs. Experimentation might be the key to fine-tune ad-hoc parameters to compute practically certain (class of) graphs.

4.3 Application on cycles

In this section, we provide a simple example of the behavior of our algorithm given two different strategies for the choice function. Before this, we just anticipates the result of the following section concerning the number of nodes developed by a general recursion tree using Algorithm 1 is $O(2^{n^2})$.

As shown in Figure 4(a), a triangulation F of the cycles is simple. The first strategy consist in adding the edges in the recursion from left to right, as shown in Figure 5. Note that, in our general algorithm, the Step 3 is inefficient, and could be removed in this case.

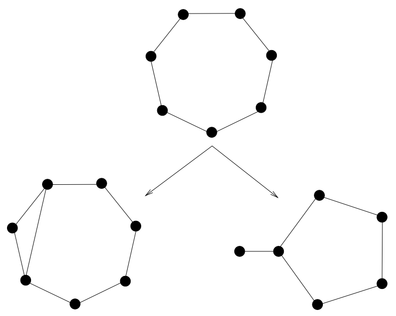


Figure 5. First step of triangulation process on C_7.

Let U_n be the number of nodes in the recursion tree for this strategy. We have the following recursion equation:

$$\begin{cases} U_{n+2} = 1 + U_{n+1} + U_n \\ U_i = 1 \qquad\qquad\qquad i \leq 3 \end{cases} \tag{9}$$

The solution of this equation is dominated by an exponential term in $O(\rho^n)$, where ρ is the golden ratio $((1+\sqrt{5})/2)$. To be more precise, we have:

$$U_n = \left(1 - \frac{\sqrt{5}}{5}\right)\rho^n + o(\rho^n) = 0.56\,\rho^n + o(\rho^n).$$

For the second strategy, we add the edge in order to complete a more central edge in the augmented clique tree, as shown in Figure 6.

Let consider now V_n as the number of nodes generated by Algorithm 3. We note that the recurrence equation depends on the parity of n leading to the following equation. For small values of n, we obtain slightly better values than by the recursion equation due to triangulated graphs that are directly generated by Step 9.

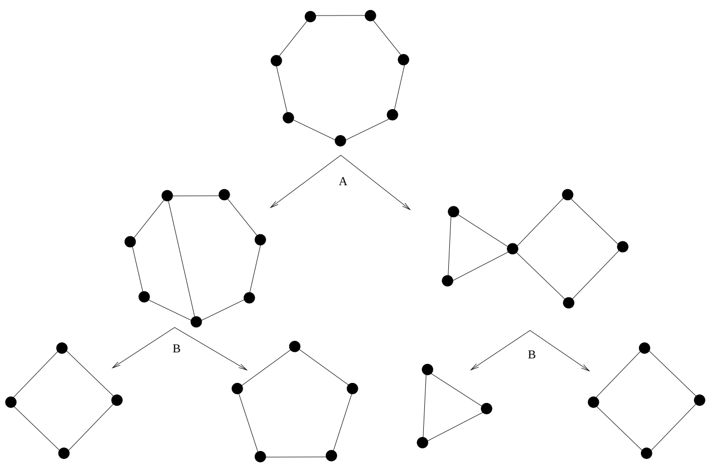


Figure 6. First steps of Algorithm 3 on C_7. We only give here two levels of recursion. We denote the recursion of type A when it corresponds to Step 9, the recursions of type B to Step 3.

$$\begin{cases} V_{2n} = 3 + 2V_n + 2V_{n+1} & \\ V_{2n+1} = 3 + V_n + 2V_{n+1} + V_{n+2} & \\ V_i = 1 & i \leq 3 \\ V_4 = 3 & \\ V_5 = 7 & \end{cases} \tag{10}$$

This recursion is satisfied by $V_n = \left\lfloor \frac{(n-1)^2}{2} \right\rfloor - 1$, for $n > 3$.

This simple example shows that the choice function is important and attention should be paied in its optimization.

5 A lower bound of the complexity

In this section, we provide a simple lower bound of the complexity of the previous algorithm directly connected to the structure of the triangulation set, viewed as a simple graph. For this, we analyze our basic Algorithm 1 in terms of number of recursion nodes in Section 5.1 and derive some lower bounds for Algorithm 3 in Section 5.2.

5.1 Tight analysis of Algorithm 1

In this section, we consider the basic algorithm for computing the chromatic polynomial of a graph using the completion strategy. Let consider that the edges to added are in the set F. We have the following result.

Lemma 4. *Let F be an edge set using labels in $\{1,\ldots,k\}$, and $n \geq k$. Let $\sum b_i^n \lambda^{(i)}$ be the chromatic polynomial of $G_n = K_n \setminus F$. Then, the value $s(n) = \sum b_i^n$ is independent of n.*

Proof. This can be shown in two ways. First, using [17], it is known that $s(n)$ counts the number of clique covers of $\overline{K_n \setminus F}$, more precisely b_i^n is the number of clique covers of $\overline{G_n}$ using i cliques. Note that, for $n \geq k$, $\overline{G_n}$ is exactly composed by $\overline{G_k}$ and $n-k$ isolated points, and the set of edges of $\overline{G_k}$ is exactly the set F. Consequently, given a clique cover of $\overline{G_k}$ using i cliques, we can associate in a canonical way an unique clique cover of $\overline{G_n}$ using $i+n-k$ cliques: add the $n-k$ K_1 corresponding to the isolated points. Indeed, this clearly provides a bijection between the set of clique covers of $\overline{G_k}$ using i cliques and the set of clique covers of $\overline{G_n}$ using $i+n-k$ cliques. Since these sets are finite, we have $b_i^k = b_{i+n-k}^n$. Since $\overline{G_n}$ contains $n-k$ isolated points, $b_i^n = 0$ for $i < n-k$. Finally, we have $s(n) = s(k)$, for $n > k$.

The second way uses the recursion tree induced by the computation of $P(G_n, \lambda)$ using of Equation 1 and stopping the recursion when the graphs are cliques. Using this paradigm $s(n)$ counts the number of leaves of this particular recursion tree. In order to conclude, we just have to note that given a recursion tree for some n, we can derive a similar (having the same structure) one for n': the idea is then to introduce the edges in the same order within both trees.

□

Let consider now the consequence of this lemma for the complexity of Algorithm 1: the way the edges are considered is not important: the number of steps will be exactly the same whatever the choice made in Step 3.

Definition 4. *Let F be an edge set using labels in $\{1,\ldots,k\}$. We define $cc(F)$ as the number of clique covers of F.*

Lemma 4 provides a way to compute this value using the chromatic polynomial. For example, it is easy to see that $cc(F) = 2$ is F is reduced to one edge, and 2^k if F is a matching of size k. In the case of a cycle, F corresponds to a complete graph minus the cycle. It can be shown that the clique cover is exactly the number of partitions of an n-set into blocks of size > 1 [18, Seq A000296] (where the first elements are 1, 0, 1, 1, 4, 11, 41, 162, 715, 3425, 17722, 98253, 580317, 3633280, 24011157, 166888165, 1216070380, 9264071767, 73600798037, 608476008122, 5224266196935, 46499892038437, 428369924118314, ...).

Theorem 3. *The number of recursion nodes developed by Algorithm 1 is exactly $2cc(\overline{G}) - 1$.*

Proof. This is a direct consequence of Lemma 4 and of the fact that the recursion tree has a special shape: all the internal nodes have 2 children, leading to the desired size.

□

In the following section, we show the impact of this bound on our algorithm and any algorithm that would be based on addition of edges and merges of non-adjacent vertices.

5.2 Lower bound for Algorithm 3

In Section 4, we have exhibited some strategy to speed up the final execution of Algorithm 3. The main point is to find the best edge in clique tree to complete in order to apply the separation step (Step 3).

Theorem 4. *Let G be graph and F a triangulation of G. Let T be the augmented clique tree of G and F. Let F_1 be the label of an edge of T. Then, the number of nodes in the recursion tree induced by Algorithm 3 is greater than $2cc(F_1) - 1$.*

Proof. The idea is to develop the recursion tree as done in the proof of Lemma 4 on F_1. In order to quantify the size of this tree, we prune it in the following way: we restrict the edges to add in Step 9 of Algorithm 3 to F_1. Furthermore, we consider the steps of separation of the clique tree (Step 3) as final steps. This reduced recursion tree can be mapped onto the recursion tree of $\overline{F_1}$ (seen as a graph) using our initial algorithm (Algorithm 1). Using Theorem 3, we obtain the desired result.

This choice to select only the edges in F_1 is valid since the studied recursion tree corresponds to the *quotient* of the effective recursion tree by the steps induced by the edges in the considered label.

□

Thus, given a triangulation of a graph, we can evaluate a lower bound of the complexity, and thus of the general computation time. The characteristic of this computation is that it is based on the same algorithm, but on smaller graphs. The computation of this lower bound can be performed efficiently compared to the global time. However, this bound can be very far from the effective complexity of the algorithm since we only consider some local operations.

One initial test before using this complex one should be to consider a maximal matching of the set F_1 (of size k). As shown before, this bound is lower bounded by 2^k. This first filter would only restrict our computation to very large graph for which the separators are also large.

To conclude this section, better lower bounds should be found in order to capture very efficiently the time that will be spent in the real computation of our algorithm.

6 Experiments

In this section, we provide experimental results on several classes of graphs. The algorithms are coded in OCAML. In the following, we never consider triangulated graphs since Lemma 2 gives an efficient solution. All the computations of the thickness of triangulation parameter have been performed using the minimum degree heuristic. This only provides an upper bound of the parameter and a decomposition for this value.

6.1 Small thickness of triangulation

In this section, we examine the behavior of this algorithm on graphs having a small thickness of triangulation. In this class, we consider the cycles, the wheels, the tori and the cylinders and some circulant graphs.

Cycle: C_n, n nodes, n edges, thickness of triangulation: 1. The clique tree is a chain, as we have seen in the previous sections. Even the closed form of the chromatic polynomial is known, this constitutes a good benchmark for generic algorithms.

n	100	200	300	400	500	600	700	800	900	1000
Time (s)	0.45	2.25	5.79	12.00	20.40	32.10	48.54	70.37	97.85	129.77

Cylinders 3 times n: $Cyl(3, n)$, $3n$ nodes, $6n - 3$ edges, thickness of triangulation: 2. The clique tree is a chain. There is no real challenge here since each cycle is a separator and we can split efficiently the graph into smaller pieces. The overhead comes from the use of very long integers.

n	5	10	15	20	25	30	35	40	45	50
$\|V(G)\|$	15	30	45	60	75	90	105	120	135	150
density	.25	.13	.08	.06	.05	.04	.03	.03	.02	.02
Time (s)	0.01	0.01	0.02	0.04	0.06	0.08	0.12	0.16	0.20	0.27

n	55	100	150	200	250	300	350	400	450	500
$\|V(G)\|$	165	300	450	600	750	900	1050	1200	1350	1500
density	.02	.01	.008	.006	.005	.004	.003	.003	.002	.002
Time (s)	0.34	1.50	4.61	11.08	22.22	42.63	64.02	94.72	136.60	188.63

Cylinders 4 times n: $Cyl(4, n)$, $4n$ nodes, $8n - 4$ edges, thickness of triangulation: 4. The clique tree is a chain with extremities having two leaves.

n	5	10	15	20	25
$\|V(G)\|$	20	40	60	80	100
density	.18	.09	.06	.04	.03
Time (s)	0.09	2.20	9.59	66.16	129.97

The thickness of triangulation increases a lot with the size of the cycle in the cylinder. The corresponding clique tree is more and more compact. All these tends to let more complex the chromatic polynomial.

Grids 2 times n: $M(2, n)$, $2n$ nodes, $3n - 2$ edges, thickness of triangulation: 1. Same comment as for $Cyl(3, n)$.

n	100	200	300	400	500	600	700	800	900
Time (s)	0.64	4.22	13.88	34.35	70.55	138.67	214.07	324.66	469.44

Grids 3 times n: $M(3, n)$, $3n$ nodes, $5n - 3$ edges, thickness of triangulation: 2. In this example, we see the influence of the choice of the first edge in the triangulation process (Step 9 in Algorithm 3). The augmented clique tree in this case is the same as for the cylinders $Cyl(4, n)$. However, the choice made in this example was to take the most central edge of the augmented clique tree. For even n, this edge has load 2, whereas it has load 1 for odd n.

n	5	10	15	20	25	30	35	40	45	50	55
Time (s)	0.03	0.07	0.20	0.47	0.43	1.49	1.58	3.02	2.40	4.77	5.39

n	60	65	70	75	80	85	90	95	100	150	200
Time (s)	10.18	5.2	13.94	11.91	23.01	12.37	26.21	28.08	40.94	159.61	563.78

Grids 4 times n: $M(4, n)$, $4n$ nodes, $7n - 4$ edges, thickness of triangulation: 5. The clique tree is more compact than before. Here again, the choice of the first edge is crucial to obtain good performance. As example, for $n = 8$, the first edge of the clique tree to be completed has thickness 5, inducing a 32 overhead, whereas for $n = 9$, the first edge has thickness 3, inducing only an 8 overhead.

n	4	5	6	7	8	9	10	11	12
Time (s)	0.05	0.07	0.42	0.63	1.63	0.91	4.76	6.60	13.14

n	13	14	15	16	17	18	19	20	25
Time (s)	7.83	35.62	38.44	96.40	15.29	149.33	185.65	383.59	250.97

Miscellaneous grids. Here are several results on different grids $M(n, m)$.

(n, m)	(2, 50)	(3, 10)	(3, 20)	(4, 10)	(5, 5)	(5, 7)	(6, 6)
#vertices	100	30	60	40	25	35	36
#edges	148	47	97	66	40	58	60
density	0.02	0.10	.05	.08	0.13	0.09	0.09
Thickness	1	3	3	5	8	8	14
Time (s)	0.10	0.16	1.03	5.56	0.85	12.35	74.84

Circulant graphs: $Circ(n)$, n nodes, $2n$ edges, thickness of triangulation: 4, the clique tree is a chain. In this experiment, we only present the circulant graphs with generators (+1, +2). The thickness of the other circulant is very large, and the time for processing the graphs is very large. One characteristic of this family is that all the edges in the clique tree have the same thickness.

n	10	20	30	40	50	60	70	80	90	100	120
Time (s)	0.00	0.16	0.90	3.11	8.15	19.85	42.25	103.74	209.48	424.38	1608.03

6.2 Random graphs

Many experiments on random graphs have been performed. However, we would like to emphasize several characteristics. The aim was to determine the link between the efficiency of computation, the size and the density of the graph. Each value is the mean of 3 test graphs. This clearly does not give a significant sample. However, it gives some tendency on the global behavior. As shown in the following table, the completion time of the algorithm widely depends on the density of the graph, the most difficult instances stand for medium density (around 50%).

From these experiments, we can see that the curves of the thickness of triangulation follows the time's. Some correlations with the size of the tree have been observed during other complementary investigations.

18 vertices									
density	16.56	20.48	30.50	41.39	50.98	59.91	69.93	79.30	90.63
#edges Clique Tree	14.33	13.67	10.67	9.67	7.33	6.33	5.67	4.67	3.00
diameter Clique Tree	7.33	5.33	4.67	5.00	3.33	4.00	4.00	3.33	2.00
Size Max Clique	4.67	5.33	8.33	9.33	11.67	12.67	13.33	14.33	16.00
Thickness	4.33	3.33	14.33	14.00	19.67	18.33	17.33	14.33	6.67
Time (s)	0.01	0.02	1.56	2.89	19.63	7.42	5.67	2.26	0.10

20 vertices									
density	13.33	19.65	30.00	40.53	51.05	60.70	70.53	81.75	91.05
#edges Clique Tree	16.67	14.67	11.67	9.33	7.00	6.33	5.33	4.33	3.33
diameter Clique Tree	7.00	6.67	5.67	4.00	4.00	3.67	3.00	2.67	2.00
Size Max Clique	4.33	6.33	9.33	11.67	13.67	14.67	15.33	16.67	17.33
Thickness	3.00	7.33	19.00	27.00	25.33	27.00	22.67	16.00	10.00
Time (s)	0.00	0.09	12.93	169.48	186.51	130.10	57.91	32.57	0.59

21 vertices									
density	13.02	20.48	30.79	41.43	51.75	60.63	70.63	81.59	90.79
#edges Clique Tree	17.33	15.00	12.00	9.33	7.67	7.33	5.67	4.33	3.33
diameter Clique Tree	7.33	6.67	6.00	5.33	4.00	4.67	3.00	2.33	2.00
Size Max Clique	4.33	7.00	10.00	12.67	14.67	14.67	16.33	17.67	18.67
Thickness	2.00	9.67	20.67	35.33	28.33	30.33	26.67	18.67	12.33
Time (s)	0.01	0.36	59.43	1943[1]	1150	478.33	120.52	55.63	2.02

22 vertices									
density	13.28	20.20	30.30	40.98	51.80	60.61	71.43	81.96	91.05
#edges Clique Tree	18.33	16.33	12.67	9.67	7.33	7.00	5.67	4.67	3.33
diameter Clique Tree	6.33	7.33	6.00	5.33	4.33	4.33	3.00	2.67	2.33
Size Max Clique	4.33	6.67	10.33	13.33	15.67	16.00	17.33	18.33	19.67
Thickness	3.33	11.00	19.67	36.00	34.00	37.00	27.67	21.33	13.00
Time (s)	0.03	0.66	193.85	2080[1]	****[2]	****	340.67	100.35	3.34

25 vertices									
density	12.78	20.56	30.67	41.56	52.44	61.44	72.00	81.33	90.33
#edges Clique Tree	20.67	16.67	13.33	9.67	8.00	7.33	6.00	4.67	4.00
diameter Clique Tree	7.33	6.67	6.00	4.33	4.67	3.67	3.67	3.00	2.67
Size Max Clique	5.33	8.67	12.67	16.33	18.00	18.67	20.00	21.33	22.00
Thickness	8.00	15.00	32.00	55.00	55.33	52.00	42.00	32.33	18.00
Time (s)	0.54	14.32	840.5[1]	****	****	****	****	670[1]	33.62

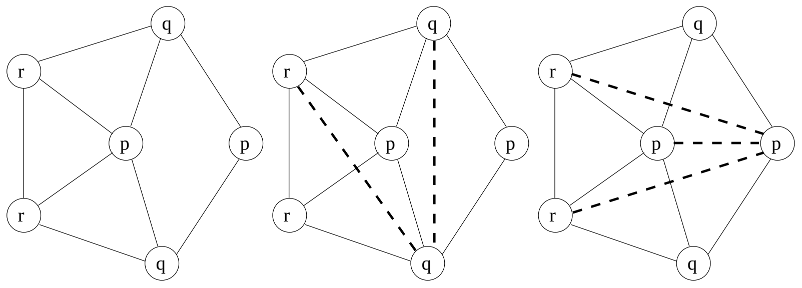
[1] One of the graphs leads to a memory overflow.

[2] Two or more graphs of this experiment lead to a memory overflow.

7 Conclusion

This work constitutes a first approach of the computation of the chromatic polynomial using the triangulation theory of the graphs. Doing this, we have enlightened some correlations between a new parameter of triangulation, namely the thickness of triangulation, and the computation time of our algorithm. Some non-trivial lower bounds, using the computation of chromatic polynomials on small graphs have been shown.

Many questions arise from this work. First of all, is this new parameter can be optimized independently from the other classical parameters of triangulation, as the minimum fill-in. In the example of Figure 7, we show that both parameters are not optimal for the same triangulations. In this example, only two minimal triangulations are possible (up to symmetry). The minimal degree heuristic finds the triangulation that minimizes the fill-in.



(a) Graph (b) Fill-In = 27; Thickness = 18 (c) Fill-In = 28; Thickness = 16

Figure 7. A graph for which the minimum fill-in and the thickness of triangulation does not exist for the same triangulations. Each circle denotes a complete graph and each edge represents a complete bipartite graph, and $p = 2$, $q = 3$ and $r = 6$.

Here are some simple questions that are still open on this new parameter:

– Does the determination of the thickness of triangulation is NPO-complete?
– Does there exists some simple approximation schemes?
– What are the connections with the classical graph parameters, such as the treewidth?
– Are the graphs with small thickness easy to recognize?

Concerning the computation of the chromatic polynomial itself, this work constitutes a first step in the elaboration of a new heuristic. Studies on the choice function have to be performed and analyzed. Another direction of experimentation could be to use the two opposite approaches for this computation.

First, use the remove and contract edges paradigm (Equation 2) in order to break large cliques, inducing large thickness, then use our approach when the triangulation induces a small enough thickness.

Acknowledgments

The authors thank Ioan Todinca for the counterexample of Figure 7.

References

1. B. Aspvall and P. Heggernes. Finding minimum height elimination trees for interval graphs in polynomial time. *BIT*, 34(4):484–509, 1994.
2. C. Berge. *Graphs and Hypergraphs*. North Holland, Amsterdam, 1973.
3. A. Berry, P. Heggernes, and G. Simonet. The minimum degree heuristic and the minimal triangulation process. In H.L. Bodlaender, editor, *Graph Theoretical Concepts in Computer Science (WG) 2003*, volume 2880 of *Lecture Notes in Computer Science*, pages 58–70, October 2003.
4. G.D. Birkhoff and D.C. Lewis. Chromatic polynomials. *Transactions of the American Mathematical Society*, 60:355–451, 1946.
5. N. Chandrasekharan, C.E.V. Madhavan, and R. Laskar. Chromatic polynomials of chordal graphs. *Congressus Numerantium*, 61:133–142, 1988.
6. S-C. Chang. Exact chromatic polynomials for toroidal chain of complete graphs. *Physica A*, 313:397–426, 2002.
7. F.M. Dong, K.L. Tep, K.M. Koh, and M.D. Hendy. Non-chordal graphs having integral-root chromatic polynomial II. *Discrete Mathematics*, 245:247–253, 2002.
8. B. Eisenberg. Generalized lower bonds for the absolute values of the coefficients of chromatic polynomials. In *Recent Trends in Graph Theory*, volume 186 of *Lecture Notes in Mathematics*, pages 85–94. Springer-Verlag, 1971.
9. P. Galinier, M. Habib, and C. Paul. Chordal graphs and their clique graph. In M. Nagl, editor, *Graph Theoretical Concepts in Computer Science (WG)*, volume 1017 of *Lecture Notes in Computer Science*, pages 358–371, Aachen, Germany, June 1995. 21st Internationnal Workshop WG'95, Springer.
10. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.
11. G. Haggard. Computing chromatic polynomials of large graphs I. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 13:175–186, 1993.
12. G. Haggard. Using thresholds to compute chromatic polynomials. *Ars Combinatoria*, 58:85–95, 2001.
13. G. Haggard and T.R. Mathies. Note on the computation of chromatic polynomials. *Discrete Mathematics*, 199:227–231, 1999.
14. B. Jackson. Zeros of chromatic and flow polynomials of graphs. *Journal of Geometry*, 76:95–109, 2003.
15. A. Natanzon, R. Shamir, and R. Sharan. A polynomial approximation algorithm for the minimum fill-in problem. In ACM, editor, *ACM Symposium On Theory of Computing*, pages 41–47, New York, NY, USA, 1998. ACM Press.
16. J. Oxley and D. Welsh. Chromatic, flow and reliability polynomials: the complexity of their coefficients. *Combinatorics, Probability and Computing*, 11:403–426, 2002.

17. D.R Shier and N. Chandrasekharan. Algorithms for computing the chromatic polynomial. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 4:213–222, 1988.
18. N.J.A. Sloane. On-line encyclopedia of integer sequences. `http://www.research.att.com/~njas/sequences/index.html`.
19. M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 2(1):77–79, March 1981.

Appendix

A Proof of Theorem 1

For the sake of simplicity and as far as we treat undirected graphs, we will consider that (a, b) and (b, a) represent the same edge.
Let

- $G = (V, E)$ be a chordal graph.
- $T = (\mathcal{V}, \mathcal{E})$ be a clique tree of G.
- $e = (i, j) \in E$.
- $G/e = (V/e, E/e)$ be the graph obtained by contracting the edge e in G:
  - $V/e = (V - \{i, j\}) \cup \{\mu\}$
  - $E/e = \{(a, b) \in E \mid \{a, b\} \subseteq (V - \{i, j\})\}$
    $\cup (\{(\mu, a) \mid ((i, a) \in E) \vee ((j, a) \in E)\} - \{\mu, \mu\})$

(here, μ represent the "new" vertex resulting in the contraction of i and j).
Let $C^{\Delta} = (\mathcal{V}^{\Delta}, \mathcal{E}^{\Delta})$ where:

- $\Delta(v) = \begin{cases} V & \text{if } i \notin V \wedge j \notin V \\ (V - \{i, j\}) \cup \{\mu\} & \text{otherwise} \end{cases}$
- $\mathcal{V}^{\Delta} = \{\Delta(V) \mid v \in \mathcal{V}\}$
- $\mathcal{E}^{\Delta} = \{(\Delta(a), \Delta(b)) \mid (a, b) \in \mathcal{E})\}$

Let $C' = (\mathcal{V}', \mathcal{E}')$ be the graph obtained by contracting an edge (e_1, e_2) in C^{Δ} each time that $e_1 \subseteq e_2$. We want to show that C' is a clique tree of G/e. The proof is decomposed in two steps:

1. We will first show that for C^{Δ} the following properties hold:
   a. it's a tree.
   b. $\mathcal{V}^{\Delta}$ is composed only of cliques of G/e and contain all the maximal cliques.
   c. $\forall v \in V/e$, the set of elements of $\mathcal{V}^{\Delta}$ containing v is a sub-tree of C^{Δ}.
2. Then, we will show that C' is actually a clique tree of G/e, that is to say that:
   a. it's a tree.
   b. C' contains only the maximal cliques of G/e. In fact, we show that any non-maximal clique of G/e in $\mathcal{V}^{\Delta}$ is eliminated in $\mathcal{V}'$, based on Property 1.a.
   c. $\forall v \in V/e$, the set of elements of $\mathcal{V}'$ containing v is a sub-tree of C'.

We will first demonstrate some useful lemma.

Lemma 5. *if α is a clique of G/e and $\mu \notin \alpha$, then α is clique of G.*

Proof. It's obvious because edges of $\mathcal{E}$ and $\mathcal{E}^{\Delta}$ are the same as far as neither i nor j nor μ are concerned.

□

Lemma 6. *if $\alpha \cup \{\mu\}$ ($\mu \notin \alpha$) is a clique of G/e, then $\alpha \cup \{i\}$ or $\alpha \cup \{j\}$ are cliques of G.*

Proof. Let α be a set of element of V/e with $\mu \notin \alpha$ and such that $\alpha \cup \{\mu\}$ is a clique of G/e. By Lemma 5, we know that α is a clique of G. Suppose that neither $\alpha \cup \{i\}$, nor $\alpha \cup \{j\}$ are cliques of G. We can infer that their exists u and v in α such that:

– $(i, u) \notin E$ ($\alpha \cup \{i\}$ is not a clique of G)
  but $(j, u) \in E$ (else $(\mu, u) \notin E/e$)
– $(j, v) \notin E$ ($\alpha \cup \{j\}$ is not a clique of G)
  but $(i, v) \in E$ (else $(\mu, v) \notin E/e$)
– $u \neq v$ (because for example, $(i, u) \in E$ but $(i, v) \notin E$).
– $(u, v) \in E$ (u and v are in α, clique of G).

It leads that the path $ijuv$ in G is a cycle of length 4 (without chords). But it's absurd as G is chordal.

□

We can now go on with the main proof:

1. a. C^{Δ} **is a tree**: derived from definition.
   b. $\mathcal{V}^{\Delta}$ **is composed only of cliques of** G/e **and contain all the maximal cliques:**
   – $\forall\ v \in \mathcal{V}$, $\Delta(v)$ **is a clique of** G/e **(**$\mathcal{V}^{\Delta}$ **contains only cliques of** G/e**)**:
     Let $v \in \mathcal{V}$, v is a clique of G, then by cases analysis:
     • $j \notin v$ and $i \notin v$: then $\Delta(v) = v$ and v is always a clique of G/e.
     • $j \in v$ or $i \in v$: $\Delta(v) = (v - \{i, j\}) \cup \{\mu\}$, but $v - \{i, j\}$ is clearly a clique of G/e and, as v is a clique of G and as $\{(\mu, a) \mid ((i, a) \in E) \vee ((j, a) \in E)\} \subseteq E/e$, v is a clique of G/e.
   – $\mathcal{V}^{\Delta}$ **contains all the maximal cliques of** G/e**:**
     We suppose that there exists α a maximal clique of G/e which is not in $\mathcal{V}^{\Delta}$. We will show that it leads always to a contradiction. We have two cases:
     i. $\mu \notin \alpha$: by Lemma 5, α is a clique of G. We also have two cases:
        • α is maximal: then $\alpha \in \mathcal{V}$ and $\Delta(\alpha) = \alpha$ is in $\mathcal{V}^{\Delta}$, absurd.
        • α is not maximal: there exists a maximal clique β such that $\alpha \subset \beta$ (and $\beta \in \mathcal{V}$):
          ∗ if $\beta = \alpha \cup \{j\}$ or $\beta = \alpha \cup \{i\}$, then $\Delta(\beta) = \alpha \cup \{\mu\}$. But, as $\mu \notin \alpha$, $\Delta(\beta)$ is a clique of G/e such that $\alpha \subset \Delta(\beta)$. It contradict the maximality of α.
          ∗ otherwise, there exists a vertex $v \in V$, different from i and j, such that $\beta = \alpha \cup \{v\} \cup \gamma$ for some γ. But then, $\Delta(\beta) = \alpha \cup \{v\} \cup \gamma'$ for some γ'. This comes in contradiction with the hypothesis of α being maximal.
     ii. $\mu \in \alpha$: let $\alpha^{-} = \alpha - \mu$, by Lemma 6, we know that either $\alpha^{-} \cup \{i\}$ or $\alpha^{-} \cup \{j\}$ is a clique of G. Let's call α' this clique. There are two distinct cases:

- α' is maximal: then $\alpha' \in \mathcal{V}$ and $\Delta(\alpha') = \alpha$ is in $\mathcal{V}^{\Delta}$, absurd.
- α' is not maximal: there exists a maximal clique β such that $\alpha' \subset \beta$ (and $\beta \in \mathcal{V}$):
  - ∗ if $\beta = \alpha' \cup \{j\}$ or $\beta = \alpha' \cup \{i\}$, then $\Delta(\beta) = \alpha \cup \{\mu\} = \alpha$. It's absurd as $\alpha \notin \mathcal{V}^{\Delta}$.
  - ∗ otherwise, there exists a vertex $v \in V$, different from i and j, such that $\beta = \alpha' \cup \{v\} \cup \gamma$ for some γ. But then, $\Delta(\beta) = \alpha \cup \{v\} \cup \gamma'$ for some γ'. It is absurd as α is maximal.

c. $\forall\, v \in V/e$**, the set of elements of** $\mathcal{V}^{\Delta}$ **containing** v **is a sub-tree of** C^{Δ}**.**
Let's remember that this property is true for C. Let v be an element of V/e. if $v \neq \mu$, as the property is true for that vertex for C, it is necessarily true for C^{Δ} as we only change all the occurrences of j and i to μ. For $v = \mu$, μ occurs in all the vertices of C^{Δ} resulting of vertices of C containing i or j. But, in C, for i and j, the set of elements of $\mathcal{V}$ containing them forms two sub-trees. And, as $e \in E$, e is in a maximal clique of G, these two sub-tree are connected by at least one vertex of C. Consequently, in the renaming, the set of elements of $\mathcal{V}^{\Delta}$ contains μ is a sub-tree of C^{Δ}.

2. a. C' **is a tree:** derived from definition.

b. **We want to show that the contractions done for obtaining** C' **suppress all and only the non-maximal cliques of** $\mathcal{V}'$ **from** G/e**.**
We know that C^{Δ} contains only cliques of G/e in $\mathcal{V}'$, and we know that there is in particular the maximal cliques of G/e. For any clique c of $\mathcal{V}'$ which is not maximal, we know that there is a maximal clique α in $\mathcal{V}'$ which includes it. And by Property 1.b, we know that there exists a path in $\mathcal{C}'$ from c to α, such that all c is included in all the clique of this path. Then, by contraction, this clique will be suppressed. And of course, no maximal clique can be suppressed.

c. $\forall\, v \in V/e$**, the set of elements of** $\mathcal{V}'$ **containing** v **is a sub-tree of** C'**.**
We have showed that this property is true for C^{Δ}. The operation from C^{Δ} to C' is to contract edges when the vertex at each sides is included one in the other (and we keep the bigger one). This operation clearly couldn't break the property.

All these elements complete the proof of Theorem 1.