AUSTRALASIAN JOURNAL OF COMBINATORICS
Volume **75(2)** (2019), Pages 210–222

On an algorithm for comparing the chromatic symmetric functions of trees

S. Heil C. Ji

Department of Mathematics
Washington University in St. Louis
St. Louis, MO 63130, U.S.A.
sheil@wustl.edu caleb.ji@wustl.edu

Abstract

It is a long-standing question of Stanley whether or not the chromatic symmetric function (CSF) distinguishes unrooted trees. Previously, the best computational result proved that it distinguishes all trees with at most 25 vertices (K. Russell, 2014, http://jlmartin.faculty.ku.edu/~jlmartin/CSF/). In this paper, we present a novel probabilistic algorithm which may be used to check more efficiently that the CSF distinguishes a set of trees. Applying it, we verify that the CSF distinguishes all trees with up to 29 vertices.

1 Introduction

Richard Stanley asked in [8] whether the chromatic symmetric function (CSF) distinguishes unrooted trees. Since then, it has been proven that the CSF distinguishes all trees in each of several subclasses ([1], [2], [3]). Tan ([10]) and independently Smith, Smith, and Tian ([6]) have computationally verified that the CSF distinguishes all trees on at most 23 vertices, and Russell ([5]) has shown computationally that it distinguishes all trees on at most 25 vertices.

When expressed with respect to commonly-used bases for the space of symmetric functions, the chromatic symmetric function of an arbitrary tree on n vertices contains a number of distinct terms equal to the number of partitions of n, growing super-polynomially with n. Therefore, computing the chromatic symmetric function directly requires a super-polynomial number of operations, making verification of Stanley's conjecture for trees on n vertices computationally difficult as n increases.

We present a probabilistic polynomial time algorithm for determining whether two trees S and T on n vertices have equal chromatic symmetric functions without explicitly computing the chromatic symmetric functions X_S and X_T. If in fact $X_S \neq X_T$, with probability at least $1 - \frac{1}{2^k}$ this algorithm returns a proof that $X_S \neq X_T$ in $O(kn^3)$ time, where k is a specified accuracy parameter. Using a variant of this algorithm, we verify computationally that Stanley's chromatic symmetric function

ISSN: 2202-3518

distinguishes all trees on at most 29 vertices up to isomorphism. In fact, we do so by checking that a certain truncation of the chromatic symmetric function is unique for all trees up to this size. This leads us to pose the following question, which is stronger than Stanley's original.

Question. *Does the* 3*-truncated chromatic symmetric function distinguish all unrooted trees up to isomorphism?*

The organization of this paper is as follows. In Section 2 we briefly state some standard definitions regarding chromatic symmetric polynomials. In Section 3 we give present our algorithm along with proofs of its effectiveness. Finally, in Section 4, we explain how we use our algorithm to verify the uniqueness of the chromatic symmetric function on trees with up to 29 vertices.

2 Preliminaries

We use the notation for symmetric functions found in [8].

Definition 2.1. The ith *power-sum symmetric function* is defined by

$$p_i(x_1, x_2, \dots) = \sum_{j=1}^{\infty} x_j^i.$$

For a partition $\lambda = (\lambda_1, \dots, \lambda_k) \vdash n$, one writes $p_\lambda = \prod_{i=1}^{k} p_{\lambda_i}$.

Definition 2.2. Given a graph $G = (V(G), E(G))$, a *proper coloring* of G is defined to be a mapping $\kappa : V(G) \to \mathbb{N}$ such that, for any $u, v \in V(G)$, if $uv \in E(G)$ then $\kappa(u) \neq \kappa(v)$.

Definition 2.3. (Stanley, [8], Definition 2.1) For a graph G, Stanley defined the *chromatic symmetric function* of G, denoted X_G, in [8] as follows:

$$X_G = \sum_{\kappa} \prod_{v \in V(G)} x_{\kappa(v)},$$

where $x_1, x_2, \dots$ are commuting indeterminates and the sum is taken over all proper colorings κ of G.

Additionally, when the graph G is understood, for any proper coloring κ we let $x_\kappa = \prod_{v \in V(G)} x_{\kappa(v)}$.

3 A Probabilistic Algorithm for Distinguishing Chromatic Symmetric Functions

Definition 3.1. Let G be a graph on n vertices, and let $S_{v,c}$ be the set of all proper colorings of G such that the color of vertex v is c. Then, for each vertex v of G and each color c, we define the function $Z_G^v(c)$ by

$$Z_G^v(c) = \sum_{\kappa \in S_{v,c}} \prod_{u \in G} x_{\kappa(u)}.$$

We will let $Y_G^v(c) = X_G - Z_G^v(c)$ be the sum of all terms $x_\kappa = \prod_{u \in G} x_{\kappa(u)}$ in the CSF X_G such that $\kappa(v) \neq c$.

The next two lemmas follow directly from the definitions.

Lemma 3.1. *Let G be a graph on the vertex set $\{1, 2, \ldots, n\}$, and let v be a vertex of G. Then, we have*

$$\sum_{c=1}^{\infty} Z_G^v(c) = X_G.$$

The function $Z_G^v(c)$ simplifies the task of finding the chromatic symmetric function X_G for a graph G by reducing it to cases in which the color of a certain vertex of the graph is fixed. This is particularly helpful for rooted trees, since we can reconstruct $Z_G^v(c)$ for a rooted tree given the corresponding information about its rooted subtrees.

Lemma 3.2. *Let T be a tree rooted at vertex v, and let $v_1, \ldots, v_k$ be the vertices of T adjacent to v. For each $1 \leq i \leq k$, let T_i be the connected component of $T \setminus \{v\}$ containing v_i. Then*

$$Z_T^v(c) = x_c \prod_{i=1}^{k} (X_{T_i} - Z_{T_i}^v(c)) = x_c \prod_{i=1}^{k} Y_{T_i}^v(c).$$

Lemma 3.3. *Let T be a tree on n vertices rooted at v. Then, there exists a unique n-tuple of symmetric functions, $(F_1, \ldots, F_n)$, each in the indeterminates $x_1, x_2, \ldots$, such that, for any $c \in \mathbb{N}$,*

$$Z_T^v(c) = \sum_{l=1}^{n} x_c^l F_l.$$

Proof. First, we will prove that such an n-tuple of symmetric functions must exist. We proceed by induction on n. In the base case of $n = 1$, we have $Z_T^v(c) = x_c$. For the inductive step, we use the fact that $Z_T^v(c) = x_c \prod_{i=1}^{k} (X_{T_i} - Z_{T_i}^v(c))$, by Lemma 3.2. Note that X_{T_i} is a symmetric function and by the inductive hypothesis, $Z_{T_i}^v(c)$ is a polynomial in x_c with symmetric functions as coefficients. This shows that each term in the product can be written as a polynomial in x_c with symmetric function coefficients, and taking their product along with x_c gives that $Z_T^v(c)$ is indeed of the desired form.

Uniqueness follows from the fact that the set $\{x_c, x_c^2, \ldots, x_c^n\}$ is linearly independent with respect to the symmetric functions. □

Using Lemma 3.3, we make the following definition.

Definition 3.2. For a tree T on n vertices and a vertex v of T, define the *symmetric function sequence of T rooted at v* to be the unique sequence $\text{sfs}(T, v) = (F_1, F_2, \ldots, F_n)$ of symmetric functions satisfying $Z_T^v(c) = \sum_{i=1}^{n} x_c^i F_i$.

Combining Lemmas 3.1 and 3.3 yields the following corollary.

Corollary 3.4. *If, for some vertex v of T, $sfs(T, v) = (F_1, \ldots, F_n)$, then*

$$X_T = \sum_{i=1}^{n} p_i F_i.$$

We now present a recursive algorithm for computing the chromatic symmetric function X_T of a tree T in the p-basis. In this algorithm, the tree is traversed down from its root and calls itself at each step. Lemma 3.2 and Corollary 3.4 are then used to piece together the symmetric function sequence of T.

Algorithm 3.1 COMPUTE-CSF(n-vertex tree T)

$v \leftarrow$ arbitrary vertex of T
$(F_1, F_2, \ldots, F_n) \leftarrow$ COMPUTE-SFS(T, v)
return $p_1F_1 + p_2F_2 + \cdots + p_nF_n$

Algorithm 3.2 COMPUTE-SFS(n-vertex tree T, vertex v)

if T is single vertex **then**
    return (1)
end if
$v_1, \ldots, v_k \leftarrow$ vertices adjacent to v in T
for $i = 1$ to k **do**
    $T_i \leftarrow$ connected component of T rooted at v_i
end for
$F_1 \leftarrow 1$
for $i = 2$ to n **do**
    $F_i \leftarrow 0$
end for
$d \leftarrow 1$ // d is the degree of the polynomial we create
for $i = 1$ to k **do**
    for $j = 1$ to d **do**
        $H_j \leftarrow F_j$
    end for
    $G_1, \ldots, G_m \leftarrow$ COMPUTE-SFS(T_i, v_i) // $Z_{T_i}(c) = G_1x_c + \cdots + G_mx_c^m$
    // $X_{T_i} - Z_{T_i}(c) = \tilde{G}_0 + \tilde{G}_1x_c + \cdots + \tilde{G}_mx_c^m$
    $\tilde{G}_0 \leftarrow p_1G_1 + p_2G_2 + \cdots + p_mG_m$ // compute the CSF X_{T_i}
    for $j = 1$ to m **do**
        $\tilde{G}_j \leftarrow -G_j$
    end for
    for $j = 1$ to $m + d$ **do**
        $F_j \leftarrow 0$
    end for

// multiply the old polynomial $(F_1x_c+F_2x_c^2+\cdots+F_dx_c^d)$ (stored in $H_1,\ldots,H_d$) by $(\tilde{G}_0+\tilde{G}_1x_c+\cdots+\tilde{G}_mx_c^m)$ to get a new polynomial $F_1x_c+F_2x_c^2+\cdots+F_{d+m}x_c^{d+m}$

for $j=1$ to d **do**
for $p=0$ to m **do**
$F_{j+p} \leftarrow F_{j+p} + H_j\tilde{G}_p$
end for
end for
$d \leftarrow d+m$
end for
return $(F_1, F_2, \ldots, F_n)$

Note that a call to COMPUTE-SFS(T, v) will recursively result in calls to COMPUTE-SFS for the subtree of T rooted at each vertex u, for a total of n function calls. After each COMPUTE-SFS call on a subtree of m vertices, there are m symmetric function multiplications and $m-1$ additions, followed by $(m+1)d$ multiplications and additions, for a total of at most $(m+1)(d+1)$ of each. Since $m \leq n$ and $d \leq n$, and there are n COMPUTE-SFS calls, the number of symmetric function multiplications and additions required for COMPUTE-SFS(T, v) is bounded by a polynomial in n for a tree T on n vertices.

The drawback to this recursive algorithm is the high computational cost of each symmetric function multiplication and addition. Since the chromatic symmetric function X_T of T, if represented in the p-basis, can in the worst case contain a term for each partition of n, the cost of each symmetric function multiplication and addition grows proportionally to $e^{O(\sqrt{n})}$.

To efficiently determine that the chromatic symmetric functions of a set of trees are distinct without incurring the super-polynomial cost of explicitly computing the complete chromatic symmetric function of each tree, we will define a homomorphism from the set of chromatic symmetric functions to a smaller finite set.

It follows from Theorem 2.5 of [8] that for any chromatic symmetric function X_G, there is some polynomial $F \in \mathbb{Z}[p_1, p_2, \ldots]$ such that $X_G = F(p_1, p_2, \ldots)$. An immediate corollary is that any linear combination $X = k_1X_{G_1} + \cdots + k_nX_{G_n}$ of CSFs, for integers $k_1, \ldots, k_n$, is an element of $\mathbb{Z}[p_1, p_2, \ldots]$.

Definition 3.3. Let $X \in \mathbb{Z}[p_1, p_2, \ldots]$. Then, for each $q \in \mathbb{N}$, and each infinite tuple $C = (C_1, C_2, \ldots) \in (\mathbb{Z}/q\mathbb{Z})^\infty$, define the *$C$-evaluation modulo q* of X to be the image of F under the evaluation homomorphism $\pi_{q,C} : \mathbb{Z}[p_1, p_2, \ldots] \to \mathbb{Z}/q\mathbb{Z}$ satisfying $\pi_{q,C}(F) = F(C_1, C_2, \ldots)$. We denote the C-evaluation modulo q of X by $\varphi_{q,C}(X)$.

In other words, $\pi_{q,C}$ expands X in the power basis and sets $p_i = C_i$ for each i. Note that for each $q \in \mathbb{N}$, $C \in (\mathbb{Z}/q\mathbb{Z})^\infty$, $\varphi_{q,C}$ is a homomorphism from the polynomial ring $\mathbb{Z}[p_1, p_2, \ldots]$ to the ring $\mathbb{Z}/q\mathbb{Z}$. Using the C-evaluation modulo q of the chromatic symmetric function and the fact that $\varphi_{q,C}$ is a homomorphism, we provide an analogous version of Algorithm 3.1 to compute $\varphi_{q,C}(X_T)$ for a tree T without fully computing X_T.

Algorithm 3.3 COMPUTE-EVAL-CSF(n-vertex tree T, $q \in \mathbb{N}$, $C \in (\mathbb{Z}/q\mathbb{Z})^n$)

$v \leftarrow$ arbitrary vertex of T
$(r_1, r_2, \ldots, r_n) \leftarrow$ COMPUTE-EVAL-SFS(T, v, q, C)
return $C_1r_1 + C_2r_2 + \cdots + C_nr_n \mod q$

Algorithm 3.4 COMPUTE-EVAL-SFS(n-vertex tree T, vertex v, $q \in \mathbb{N}$, $C \in (\mathbb{Z}/q\mathbb{Z})^n$)

if T is single vertex **then**
    return (1)
end if
$v_1, \ldots, v_k \leftarrow$ vertices adjacent to v in T
for $i = 1$ to k **do**
    $T_i \leftarrow$ subtree of T rooted at v_i
end for
$r_1 \leftarrow 1$
for $i = 2$ to n **do**
    $r_i \leftarrow 0$
end for
$d \leftarrow 1$ // d is the degree of the polynomial we create
for $i = 1$ to k **do**
    for $j = 1$ to d **do**
        $t_j \leftarrow r_j$
    end for
    $s_1, \ldots, s_m \leftarrow$ COMPUTE-EVAL-SFS(T_i, v_i, q, C)
    $\tilde{s_0} \leftarrow C_1s_1 + C_2s_2 + \cdots + C_ms_m \mod q$ // compute $\varphi_{q,C}(X_{T_i})$
    for $j = 1$ to m **do**
        $\tilde{s_j} \leftarrow -s_j$
    end for
    for $j = 1$ to $m + d$ **do**
        $r_j \leftarrow 0$
    end for
    for $j = 1$ to d **do**
        for $p = 0$ to m **do**
            $r_{j+p} \leftarrow r_{j+p} + t_j\tilde{s_p} \mod q$
        end for
    end for
    $d \leftarrow d + m$
end for
return $(r_1, r_2, \ldots, r_n)$

Proposition 3.5. *For a tree T on n vertices and a modulus q, Algorithm 3.3 terminates in $O(n^2(\log q)^2)$ time.*

Proof. First, the additional computation in Algorithm 3.3 after the call to Algo-

rithm 3.4 includes n multiplications and additions of $(\log q)$-bit integers, which takes $O(n(\log q)^2)$ time, so it suffices to show that 3.4 terminates in $O(n^2(\log q)^2)$ time.

Thus, we claim that for each positive integer n and for any tree T on n vertices, Algorithm 3.4 requires at most $12n^2$ addition, multiplication, and modulus operations on elements of $\mathbb{Z}/q\mathbb{Z}$.

We proceed by induction. When $n = 1$, the algorithm immediately terminates and returns (1), so the base case holds.

We now assume inductively that our claim holds for all trees on at most $n-1$ vertices, and prove that it must also hold for trees on n vertices.

Let T be a tree on n vertices, select an arbitrary root vertex v, and let $T_1, \ldots, T_k$ be the subtrees of T rooted at the children of v. Let $m_1, \ldots, m_k$ be the numbers of vertices in $T_1, \ldots, T_k$, respectively. Also, let $m_0 = 1$ for simplicity.

For each integer $1 \leq i \leq k$, let $d_i = \sum_{j=0}^{i-1} m_i$. Note that since initially $d = 1$ and m_i is added to d after iteration i, d_i is the initial value of d during the iteration of the outer loop corresponding to subtree T_i.

On iteration i, the initial for loop requires d_i operations. Then, by our inductive assumption, the call to COMPUTE-EVAL-SFS requires at most $12m_i^2$ operations. The following line includes at most m_i each of addition, multiplication, and modulus operation, for a total of $3m_i$ operations. The for loop initializing the r_i values to zero requires $d_i + m_i$ operations. The nested for loops in which $r_j s_p$ is added to r_{j+p} require at most $3d_i(m_i + 1)$ operations, since one addition, one multiplication, and one modulus operation takes place in the inner loop.

Therefore, the total number of operations performed on iteration i of the loop is at most $d_i+12m_i^2+3m_i+d_i+m_i+3d_i(m_i+1) = 12m_i^2+3m_id_i+5d_i+4m_i \leq 12m_i^2+12m_id_i$.

The number of additional steps performed in the outer loop is at most $5n$, including n each for finding the vertices adjacent to v, finding the subtrees of T rooted at these vertices, initializing the r_i values, and returning the final sequence, and the additional constant-time operations.

Taking the sum over all iterations and adding in the operations from the outer loop, the total number of operations required is at most

$$
\begin{aligned}
&5n+\sum_{i=1}^{k} 12m_i^2 + 12m_id_i \\
&\quad\leq 5n + \sum_{i=1}^{k} 12m_i^2 + \sum_{i=1}^{k} 12m_i + \sum_{i=1}^{k}\sum_{j=1}^{i-1} 12m_im_j \\
&\quad\leq 5n + \sum_{i=1}^{k} 12m_i^2 + 12m_2m_1 + \sum_{i=2}^{k-1} 12m_km_i + 12m_km_1 + \sum_{i=1}^{k}\sum_{j=1}^{i-1} 12m_im_j \\
&\quad\leq 5n + \sum_{i=1}^{k} 12m_i^2 + \sum_{i=1}^{k}\sum_{j=i+1}^{k} 12m_im_j + \sum_{i=1}^{k}\sum_{j=1}^{i-1} 12m_im_j \\
&\quad\leq 5n + \sum_{i=1}^{k}\sum_{j=1}^{k} 12m_im_j
\end{aligned}
$$

$$\begin{aligned}
&\leq 5n + 12\left(\sum_{i=1}^{k} m_i\right)^2 \\
&= 5n + 12(n-1)^2 \\
&= 5n + 12n^2 - 24n + 12 \\
&\leq 12n^2.
\end{aligned}$$

Thus, by induction, our claim is proven.

Finally, since all addition, multiplication, and modulus operations are performed on positive integers at most q, the time per operation is $O((\log q)^2)$. Therefore, as there are at most $12n^2$ operations, the total runtime of Algorithm 3.4 is $O(n^2(\log q)^2)$, which implies that Algorithm 3.3 terminates in $O(n^2(\log q)^2)$ time, as desired. □

As Algorithm 3.3 can be performed using a number of multiplications and additions of elements of $\mathbb{Z}/q\mathbb{Z}$ that is polynomial in n, if $q = O(\exp(p(n)))$ for some polynomial p, then this algorithm will terminate in polynomial time. To show that $X_S \neq X_T$ it suffices to find such a modulus q and an n-tuple $C \in (\mathbb{Z}/q\mathbb{Z})^n$ such that $\varphi_{q,C}(X_S) \neq \varphi_{q,C}(X_T)$. This leads us presently to our main theorem, after one final lemma.

Lemma 3.6. *Let q be a prime, and let $f \in (\mathbb{Z}/q\mathbb{Z})[x_1, x_2, \ldots, x_m]$ be a polynomial of degree n that is not identically zero. Then, if $C_1, C_2, \ldots, C_m$ are elements of $\mathbb{Z}/q\mathbb{Z}$ chosen uniformly at random, the probability that $f(C_1, C_2, \ldots, C_m) \equiv 0 \pmod q$ is at most $\frac{n}{q}$.*

Proof. We proceed by induction on m. The claim is true when $m = 1$ because $\mathbb{Z}/q\mathbb{Z}$ forms a field.

For the inductive step, we assume that our claim holds for polynomials in at most $m-1$ variables, for some $m \geq 2$, and we will prove that it also holds for m-variable polynomials. We group the terms of the polynomial f by powers of x_m: for some polynomials $g_0, g_1, \ldots, g_n \in (\mathbb{Z}/q\mathbb{Z})[x_1, x_2, \ldots, x_{m-1}]$, it follows that

$$f(x_1, x_2, \ldots, x_m) = \sum_{i=0}^{a} g_i(x_1, x_2, \ldots, x_{m-1}) x_m^i$$

for some $1 \leq a < n$.

There are two disjoint cases: either all the $g_i(x_1, x_2, \ldots, x_{m-1})$ always evaluate to zero, or there is some i such that $g_i(x_1, \ldots, x_{m-1})$ is not always zero. Since the degree of f is n, the degree of g_a is at most $n-a$, so the probability that $g_a \equiv 0 \pmod q$ is at most $\frac{n-a}{q}$. On the other hand, if not all the $g_i(x_1, x_2, \ldots, x_{m-1})$ evaluate to 0, then by the inductive hypothesis $\Pr[f(C_1, C_2, \ldots, C_m) \equiv 0 \pmod q] \leq \frac{a}{n}$. The probability that $f(C_1, C_2, \ldots, C_m)$ is 0 is at most the probability that every $g_i(x_1, x_2, \ldots, x_{m-1})$ is always 0 added to the probability that they are not always 0 multiplied by the probability that $f(C_1, C_2, \ldots, C_m)$ is 0. This gives $\Pr[f(C_1, C_2, \ldots, C_m) \equiv 0 \pmod q] \leq \frac{n-a}{q} + 1 \cdot \frac{a}{q} = \frac{n}{q}$.

Thus, by induction, our claim holds for all positive integers m and each value of n. □

Algorithm 3.5 SHOW-DISTINCT-CSFS(n-vertex tree S, n-vertex tree T, $k \in \mathbb{N}$)

```
q ← n^2
primeCount ← 0
while primeCount < n do
  if q is determined to be prime by trial division then
    for i = 1 to ⌊k/log_2(n)⌋ do
      for j = 1 to n do
        C_j ← random element of Z/qZ
      end for
      r_S ←COMPUTE-EVAL-CSF(S, q, C)
      r_T ←COMPUTE-EVAL-CSF(T, q, C)
      if r_S ≠ r_T then
        return 'Proved that X_S ≠ X_T.'
      end if
    end for
    primeCount ← primeCount + 1
  end if
  q ← q + 1
end while
return 'Could not prove that X_S ≠ X_T.'
```

Theorem 3.7. *For trees S and T on n vertices such that $X_S \neq X_T$ and for each $k \in \mathbb{N}$, with probability at least $1 - 2^{-k}$ Algorithm 3.5 will prove that $X_S \neq X_T$ by generating a positive integer q and a n-tuple $C = (C_1, \ldots, C_n) \in (\mathbb{Z}/q\mathbb{Z})^n$ such that $\varphi_{q,C}(X_S) \neq \varphi_{q,C}(X_T)$ in $O(n^3 k)$ time.*

Proof. First, we prove that if X_S and X_T are distinct, Algorithm 3.5 will obtain a verification that $X_S \neq X_T$ with probability at least $1 - 2^{-k}$.

Let f_S and f_T be polynomials such that $f_S(p_1, \ldots, p_n) = X_S$ and $f_T(p_1, \ldots, p_n) = X_T$, where $p_1, \ldots, p_n$ are elements of the p-basis for symmetric functions, and let $f = f_S - f_T$. By Theorem 2.5 of [8], $X_T = \sum_{U \subseteq E(T)} (-1)^{|U|} p_{\lambda(U)}$, where $E(T)$ is the set of edges of T. Since T is a tree on n vertices, $|E(T)| = n-1$, so $|\mathcal{P}(E(T))| = 2^{n-1}$. Therefore, the sum of the absolute values of the coefficients of X_T in the p-basis is at most 2^{n-1} and the same result holds for X_S. Therefore, by the Triangle Inequality, the sum of the absolute values of the coefficients of $X_S - X_T$ in the p-basis is at most 2^n.

As $f(p_1, p_2, \ldots, p_n) = X_S - X_T$, this implies that the sum of the absolute values of the coefficients of f is at most 2^n, so each coefficient of f has absolute value bounded by 2^n. As Algorithm 3.5 generates n distinct primes larger than n^2, it takes at most $\log_{n^2} 2^n = O(\frac{n}{\log n})$ primes till their product is greater than 2^n. Then one of them, say q, cannot divide all the coefficients of f. Hence, for this prime q, $f(x_1, x_2, \ldots, x_n)$ is not identically zero over $\mathbb{Z}/q\mathbb{Z}$. The algorithm generates an n-tuple C of randomly-selected residues modulo $\mathbb{Z}/q\mathbb{Z}$ and then computes $\varphi_{q,C}(X_S)$ and $\varphi_{q,C}(X_T)$.

By Lemma 3.6, since f is a polynomial in n variables, $\deg f \leq n$, and f is not identically 0 over $\mathbb{Z}/q\mathbb{Z}$ for at least one choice of q, the probability that $f(C_1, C_2, \ldots, C_n) \equiv 0 \pmod{q}$ is at most $\frac{n}{q} < \frac{n}{n^2} = \frac{1}{n}$. Therefore, with probability at least $1 - \frac{1}{n}$,

$$f(C_1, C_2, \ldots, C_n) = \varphi_{q,C}(X_S - X_T) = \varphi_{q,C}(X_S) - \varphi_{q,C}(X_T) \neq 0,$$

in which case the algorithm has shown that $\varphi_{q,C}(X_S) \neq \varphi_{q,C}(X_T)$ and hence returns that $X_S \neq X_T$.

The algorithm generates k independent n-tuples C, each leading to a proof that $X_S \neq X_T$ with probability at least $1 - \frac{1}{n}$, so the probability that it does not return $X_S \neq X_T$ after $\frac{k}{\log_2 n}$ iterations is at most $\frac{1}{n^{\frac{k}{\log_2 n}}} = \frac{1}{2^k}$, so it takes $O(\frac{k}{\log_2 n})$ iterations to achieve this desired probability. Hence, if $X_S \neq X_T$, then the algorithm will find a pair (q, C) for which $\varphi_{q,C}(X_S) \neq \varphi_{q,C}(X_T)$, showing that $X_S \neq X_T$, with probability at least $1 - 2^{-k}$.

Next, we will show that Algorithm 3.5 terminates after $O(n^3 k)$ steps.

It was proven in [4] that $\frac{x}{\log x + 2} \leq \pi(x) \leq \frac{x}{\log x - 4}$, where $\pi(x)$ is the number of primes less than x. Applying these bounds, we claim that for sufficiently large n there are at least $\frac{n}{\log n}$ primes between n^2 and $2n^2$. Note that

$$\begin{aligned}
\pi(2n^2) - \pi(n^2) &\geq \frac{2n^2}{2\log n + \log 2 + 2} - \frac{n^2}{2\log n - 4} \\
&\geq \frac{2n^2}{2\log n + 4} - \frac{n^2}{2\log n - 4} \\
&\geq \frac{2n^2(2\log n - 4) - n^2(2\log n + 4)}{4(\log n)^2 - 16} \\
&\geq \frac{2n^2 \log n - 12n^2}{4(\log n)^2 - 16} \\
&\geq \frac{n^2(\log n - 6)}{2(\log n)^2} \\
&\geq \frac{n^2}{2(\log n)^2} \\
&> \frac{n}{\log n}
\end{aligned}$$

for all $n > e^7$, as desired. Therefore, for sufficiently large n we must test at most n^2 integers for primality, each of which is at most $2n^2$. As trial division can determine whether or not n is prime in $O(\sqrt{n})$ time, this implies that our algorithm will generate the desired $\frac{n}{\log n}$ primes in $O(n^3)$ time.

For each prime q generated by our algorithm, it first generates n random elements of $\mathbb{Z}/q\mathbb{Z}$, taking $O(n \log q) = O(n \log n)$ time (since $q < 2n^2 < n^3$, $\log q \leq 3 \log n$). Then, there are 2 calls to COMPUTE-EVAL-CSF on trees on n vertices using the modulus q. By Proposition 3.5, these calls require $O(n^2(\log q)^2) = O(n^2(\log n)^2)$. Thus, each iteration of the inner loop requires $O(n \log n + n^2(\log n)^2) = O(n^2(\log n)^2)$ time.

As there are $O(\frac{k}{\log n})$ iterations for each of $O(\frac{n}{\log n})$ primes, the total runtime from the inner-loop computations (everything excluding the primality testing) is $O(n^2(\log n)^2 \cdot \frac{k}{\log n} \cdot \frac{n}{\log n}) = O(n^3 k)$.

Therefore, the total runtime of Algorithm 3.5 is $O(n^3 + n^3 k) = O(n^3 k)$. □

4 Computational Results and an Additional CSF Conjecture

We now apply a variant of Algorithm 3.3 to extend Russell's computational result in [5] that the chromatic symmetric function distinguishes all unrooted trees on at most 25 vertices up to trees on at most 29 vertices.

Let $T = (V(T), E(T))$ be a tree on n vertices. As in [8], for each subset $S \subset E(T)$ define $\lambda(S)$ to be the partition of n where the parts have sizes equal to the numbers of vertices in the connected components of the graph $(V(T), S)$.

Definition 4.1. For a tree T and a positive integer k, let $P_k(T)$ be the set of edge subsets S partitioning T into connected components each with at most k vertices:

$$P_k(T) = \{S | S \subseteq E(T), \lambda(S) = (a_1, \ldots, a_m), a_1 \leq a_2 \leq \cdots \leq a_m \leq k\} \subseteq \mathcal{P}(E(T)).$$

We then define the *k-truncated chromatic symmetric function* of T, denoted ${}_kX_T$, as follows:

$${}_kX_T = \sum_{S \in P_k(T)} (-1)^{|S|} p_{\lambda(S)}$$

Note that, by Theorem 2.5 of [8], when $k = n$ we obtain the original chromatic symmetric function X_T: ${}_kX_T = \sum_{S \subseteq E(T)} (-1)^{|S|} p_{\lambda(S)} = X_T$.

Remark 4.1. Our approach of considering only some of the possible subsets of edges of T is similar to that used by Smith, Smith, and Tian in [6]. There, they consider the terms in the chromatic symmetric function corresponding to subsets $S \subseteq E(T)$ such that $|S| \leq k$, namely all m-cuts of T where $m \leq k$. They showed in [6] that computing the terms in the chromatic symmetric function for all such subsets S where $|S| \leq 5$ suffices to prove that the chromatic symmetric function distinguishes unrooted trees on at most 24 vertices.

Here, instead of considering subsets $S \subseteq E(T)$ such that $|S| \leq k$ for a constant k, we consider subsets $S \subseteq E(T)$ such that any subset $U \subseteq V(T)$ for which $uv \in E(T) \implies uv \in S$ for all $u, v \in U$ satisfies $|U| \leq k$. In other words, the subsets we consider partition T into connected components of size at most k.

Remark 4.2. The 2-truncated chromatic symmetric function of a tree corresponds to its matching polynomial. The matching polynomial of a tree is equivalent to its characteristic polynomial and is therefore insufficient to uniquely determine it [7].

We will now use the following fact to achieve our computational result.

Observation 4.3. *Let T and T' be trees on n vertices. Then for any positive integer $k \leq n$, if $X_T = X_{T'}$, then ${}_kX_T = {}_kX_{T'}$. Thus if ${}_kX_T \neq {}_kX_{T'}$ for all pairs of non-isomorphic trees (T, T'), then all non-isomorphic trees on n vertices have unique chromatic symmetric functions.*

Remark 4.4. For $q \in \mathbb{N}$ and any $C \in (\mathbb{Z}/q\mathbb{Z})^n$ such that $C_j = 0$ for all $j > k$, $\varphi_{q,C}({}_kX_T) = \varphi_{q,C}(X_T)$.

Performing Algorithm 3.3 using an n-tuple C for which all but the first k entries are zero allows us to omit the terms $r_{k+1}, \ldots, r_n$ in the intermediate step of computing the C-evaluated symmetric function sequence (Algorithm 3.4). Furthermore, since for such an n-tuple C each term r_i with $i \geq k$ from a previous recursive call in Algorithm 3.4 only affects terms r_j, where $j \geq i \geq k$, in future calls, these remaining terms can be omitted in each call to 3.4. By avoiding the computation of these superfluous terms, we can improve the runtime of Algorithm 3.3 in this special case from $O(n^2(\log q)^2)$ to $O(nk(\log q)^2)$.

By Observation 4.3, to show that X_T distinguishes all trees on at most n vertices up to isomorphism, it suffices to show that ${}_kX_T$ distinguishes all such trees, for a constant k.

Setting $k = 3$, we obtained the following computational result:

Theorem 4.5. *The* 3*-truncated chromatic symmetric function distinguishes all trees on at most* 29 *vertices up to isomorphism. Hence, the chromatic symmetric function distinguishes all non-isomorphic trees on up through* 29 *vertices.*

Using a computer, we generated primes $q_1, q_2, \ldots, q_m$ and tuples $C_1, C_2, \ldots, C_m$ of the form $C_j = (x_j, y_j, z_j, 0, 0, \ldots)$, where $x_j, y_j, z_j \in (\mathbb{Z}/q_j\mathbb{Z})$. Then, using Keeler Russell's C++ library (which is provided in [5]) to generate all non-isomorphic trees on n vertices, we successively computed $\varphi_{q_1,C_1}(T)$ for each tree T. For the elements of each equivalence class S_r of trees T such that $\varphi_{q_1,C_1}(T) = r$, we computed $\varphi_{q_2,C_2}(T)$ and generated smaller equivalence classes of trees with each ordered pair of images under the homomorphisms φ_{q_1,C_1} and φ_{q_2,C_2}. We then repeated this process until each equivalence class contained only a single tree, which occurred after a finite sequence of homomorphisms $\varphi_{q_1,C_1}, \ldots, \varphi_{q_m,C_m}$ were applied.

The computer program we used to verify that the CSF distinguishes trees on at most 29 vertices is available at `https://github.com/VietaFan/CSFTreeConjecture`.

This computational result leads to the following question:

Question. *Does the* 3*-truncated chromatic symmetric function distinguish all unrooted trees up to isomorphism?*

An affirmative answer to this question would also answer Stanley's question on the uniqueness of chromatic symmetric functions for trees.

Remark 4.6. The size of the input data is the source of the barrier to going past 29 vertices. We expect that a computer with more space will be able to cross this threshold.

Acknowledgements

We would like to thank Dr. John Shareshian for many helpful discussions on the theory of chromatic symmetric functions that led to this paper.

We would also like to thank Keeler Russell for making the code he used to verify that the chromatic symmetric function distinguishes trees on at most 25 vertices freely available. His library for sequentially generating all non-isomorphic free trees on n vertices, previously available on Github, was particularly useful for our verification that the CSF distinguishes all trees on at most 29 vertices.

Finally, we thank the anonymous referees for making several helpful comments and suggestions.

References

[1] J. Aliste-Prieto and J. Zamora, Proper caterpillars are distinguished by their chromatic symmetric function, *Discrete Math.* 315 (2014), 158–164.

[2] J. Martin, M. Morin and J. Wagner, On distinguishing trees by their chromatic symmetric functions, *J. Combin. Theory Ser. A* 115 (2008), 237–253.

[3] R. Orellana and G. Scott, Graphs with Equal Chromatic Symmetric Functions, *Discrete Math.* 320 (2014), 1–14.

[4] B. Rosser, Explicit Bounds for Some Functions of Prime Numbers, *Amer. J. Math.* 63 (1) (1941), 211–232.

[5] K. Russell, Searching for trees with the same chromatic symmetric function, `http://jlmartin.faculty.ku.edu/~jlmartin/CSF/` .

[6] I. Smith, Z. Smith, and P. Tian, Symmetric Chromatic Polynomial of Trees, arXiv:1505.01889v2 [math.CO].

[7] D. Spielman, Matching Polynomials of graphs. Lecture notes (2015), `http://www.cs.yale.edu/homes/spielman/561/lect25-15.pdf` .

[8] R. Stanley, A Symmetric Function Generalization of the Chromatic Polynomial of a Graph, *Adv. in Math.* 111 (1995), 166–194.

[9] R. Stanley, Enumerative Combinatorics, Vol. 2, Cambridge University Press, Cambridge, 1999.

[10] L. Tan, Personal communication.

(Received 19 Dec 2018; revised 6 Aug 2019)