Efficient Computation of the Characteristic Polynomial of a Tree and Related Tasks

Martin Fürer*

Department of Computer Science and Engineering
Pennsylvania State University
University Park, PA 16802, USA
Visiting: ALGO EPFL
1015 Lausanne
Switzerland
and
Institut für Mathemtik
Universität Zürich
CH-8057 Zrich
Switzerland
furer@cse.psu.edu
http://cse.psu.edu/~furer

Abstract. An $O(n \log^2 n)$ algorithm is presented to compute the characteristic polynomial of a tree on n vertices improving on the previously best quadratic time. With the same running time, the algorithm can be generalized in two directions. The algoritm is a counting algorithm, and the same ideas can be used to count other objects. For example, one can count the number of independent sets of all possible sizes simultaneously with the same running time. These counting algorithms not only work for trees, but can be extended to arbitrary graphs of bounded tree-width.

Keywords: characteristic polynomial, counting matchings, counting independent sets, bounded tree-width, efficient algorithms.

1 Introduction

It is easy to find a maximum independent set or a maximum matching in a tree in linear time. The size of the latter determines the rank of the adjacency matrix and therefore the number of trailing zero coefficients of the characteristic polynomial. Still in linear time, one can also compute the number of the maximum matchings in a tree, and therefore determine the lowest non-trivial coefficient of the characteristic polynomial. But if the goal is to compute all coefficients of the characteristic polynomial or count the number of independent sets of size r for all possible values of r simultaneously, it has been believed that the necessary time would increase by a factor of n. We show that an increase by a factor of only $O(\log^2 n)$ is sufficient for such tasks.

* Research supported in part by NSF Grant CCF-0728921.

A. Fiat and P. Sanders (Eds.): ESA 2009, LNCS 5757, pp. 11–22, 2009.

For any graph with adjacency matrix A, elementary considerations of the characteristic polynomial

$$\chi(A;\lambda) = \det(\lambda I - A) = \sum_{i=0}^{n} c_i \lambda^{n-i}$$

show the well known result that

$$c_r = \sum_{\sigma} \operatorname{sgn}(\sigma)$$

where σ ranges over all directed cycle packings covering r vertices, i.e., permutations with with exactly $n-r$ fixed points and $A_{i\sigma(i)} = 1$ whenever i is not a fixed point of σ.

We consider undirected graphs without self-loops. Every single edge $\{u, v\}$ represents one directed cycle $(u, v), (v, u)$, while every undirected cycle represents two directed cycles (one in each direction). Naturally, in a tree the only directed cycles are those corresponding to single edges. Thus the number of cycle packings covering $2r$ vertices in a tree is the number of matchings of size r. We call them *r-matchings*.

For a tree or forest, the previous observation implies $c_{2r+1} = 0$ and

$$c_{2r} = (-1)^r \#r\text{-matchings}$$

(see, e.g., [1, p. 49]).

An early algorithm [2] for the characteristic polynomial of a tree runs in time $O(n^3)$. More complicated algorithms are needed for general graphs, but the time can even be improved. Computing the characteristic polynomial of an arbitrary real matrix has actually the same algebraic complexity as matrix multiplications [3] (see [4, Chap. 16]). Thus, with the fastest known algorithm, it can be computed in time $O(n^{2.376})$ [5]. All running times are based on the algebraic complexity measure where every arithmetic operation counts as one step.

As adjacency matrices of trees are sparse and have special structural properties, one could hope for faster algorithms. Indeed, there are algorithms to compute the determinant of the adjacency matrix of a tree in linear time [6] and the characteristic polynomial in time $O(n^2)$ [7] (also later rediscovered [8]).

A main result of this paper is to improve the running time for the computation of the characteristic polynomial of a tree to $O(n \log^2 n)$ using a novel divide and conquer approach. Computing the characteristic polynomial of a tree is equivalent to counting the number r-matchings simultaneously for all r. Thus, it is not astonishing that our new method can be applied to a wider class of simultaneous counting problems.

Many computational approaches use self-reduction. A problem is solved by solving a set of smaller problems of the same type. Quite often these smaller problems are not completely independent, but actually have a fair amount of common substructures. Our novel divide and conquer approach aims at using this similarity and solving the collection of smaller problems together with significant savings.

Our simultaneous counting method is not restricted to trees, but extends in a natural way to graphs of bounded tree-width k. For constant k, we obtain several $O(n \log^2 n)$ time simultaneous counting algorithms even for problems that are NP hard without a bound on the tree-width. The time improvement is always a factor of $\Omega(n/\log^2 n)$ compared to algorithms based on traditional techniques.

The area of algorithms has a significant branch dealing with parameterized complexity. The complexity of problems is not just studied depending on the size n of an instance, but together with an additional parameter k. The idea is that even for large n an instance of a difficult problem might still be easy if its parameter k is small. A problem is fixed-parameter tractable if it can be solved in time $O(f(k)n^c)$ for an arbitrary function f and a constant c. For example, the NP-complete Independent Set problem is solvable in time $O(2^k n)$ for graphs of tree-width k. Our result implies that with just a factor of $O(\log^2 n)$ more time, we can simultaneously count the number of independent sets of every size r in graphs of tree-width k.

2 Computing the Characteristic Polynomial

When we count objects like r-matchings (i.e., matchings of size r) it is convenient to encode them by a generating polynomial.

Definition 1. *With a_r being the number of r-matchings*

$$f_M(G;x) = \sum_{r=0}^{\lceil n/2 \rceil} a_r x^r$$

is the matching generating polynomial (see e.g. [9]).

This greatly simplifies the description of the algorithms, as the polynomial multiplication is actually an important computational step.

Similarly, we could define the generating polynomials for independent sets, vertex covers, dominating sets, and so on. These polynomials are defined for all graphs, and some might well be worth studying for their structural properties.

Definition 2. *With b_r being the number of independent sets of size r*

$$f_I(G;x) = \sum_{r=0}^{n} b_r x^r$$

is the independent set generating polynomial.

For trees (and forests), but not for general graphs, there is a well known strong relationship between the matching generating polynomial

$$f_M(G;x) = \sum_{r=0}^{\lceil n/2 \rceil} a_r x^r$$

Algorithm Characteristic-Polynomial:

Input: A tree $T = (V, E)$ with $|V| = n$.

Output: The coefficients $c_0, c_1, \ldots, c_n$ of the characteristic polynomial $\chi(A; \lambda) = \det(\lambda I - A) = \sum_{i=1}^{n} c_i \lambda^{n-i}$, where A is the adjacency matrix of T.

Comment: Use the fact that for trees $c_{2r+1} = 0$ and $c_{2r} = (-1)^r \#$ r-matchings.

$(a_0, \ldots, a_{\lfloor n/2 \rfloor}) =$ Matchings(T)
for $r = 1$ **to** $\lceil n/2 \rceil$ **do**
  $c_{2r-1} = 0$
for $r = 0$ **to** $\lfloor n/2 \rfloor$ **do**
  $c_{2r} = (-1)^r a_r$
Return $(c_0, \ldots, c_n)$

Fig. 1. The algorithm Characteristic-Polynomial

Algorithm Matchings:

Input: A tree $T = (V, E)$ with $|V| = n$.

Output: The vector $(a_0, a_1, \ldots, a_{\lfloor n/2 \rfloor})$ where a_r is the number of r-matchings in T.

$(a_0 + a_1 x + \cdots + a_{\lfloor n/2 \rfloor} x^{\lfloor n/2 \rfloor}) =$ Restricted-Matchings$(T, \emptyset)$
Return $(a_0, \ldots, a_{\lfloor n/2 \rfloor})$

Fig. 2. The algorithm Matchings

and the characteristic polynomial $\chi(G; \lambda)$, namely

$$\chi(G; \lambda) = \lambda^n f_M(G; -\lambda^{-2})$$

This is a direct consequence of the characterization of the coefficients c_r of the characteristic polynomial for forests. $c_{2r+1} = 0$ and $c_{2r} = (-1)^r \#r$-matchings (see, e.g., [1, p. 49]).

Thus, we could actually have used the characteristic polynomial directly in our algorithms. But besides the waste of half the coefficients (being 0), the use of the matching generating polynomial is more natural. It also emphasizes that no hidden algebraic properties of the characteristic polynomial are used, and algorithms immediately generalize to counting other things like independent sets.

We describe the algorithm to compute the characteristic polynomial in detail using pseudo-code. The algorithm Characteristic-Polynomial (Figure 1) inputs a tree T and just outputs the coefficients of the characteristic polynomial after receiving the coefficients of the matching generating polynomial $f_M(T, x)$ from the algorithm Matching. The algorithm Matching itself (Figure 2) inputs the tree T and outputs the coefficients of the matching generating polynomial $f_M(T, x)$, after calling the recursive procedure Restricted-Matchings.

The actual work is done in the recursive procedure Restricted-Matchings (Figure 3). Besides the tree T, it receives a small subset U of the vertices as input. Its task is not only to compute the matching generating polynomial for T, but for the subgraphs of $T = (V, E)$ induced by $V \setminus W$ for all $W \subseteq U$. Naturally, these subgraphs of T are forests.

A minor feature of the procedure Restricted-Matchings (Figure 3) is the use of approximate sizes of graphs. The approximate size of a graph with n vertices is defined to be $2^{\lfloor \lg n \rfloor}$ where lg is the logarithm to the base 2. The procedure Restricted-Matchings repeatedly selects a pair of approximately smallest trees, i.e., trees of minimal approximate size. Approximately smallest trees are as good a smallest trees, but there is no need to sort the trees by size. A bucket for each approximate size is sufficient.

Procedure Restricted-Matchings:

Input: A tree $T = (V, E)$ and a subset $U \subseteq V$.

Output: The function $\mathbf{f}$ from the powerset of U into the polynomials $\mathbb{Z}[x]$ where for every subset $W \subseteq U$, $\mathbf{f}(W) = a_0^W + a_1^W x \ldots, a_{\lfloor n/2 \rfloor}^W x^{\lfloor n/2 \rfloor}$ with a_r^W being the number of r-matchings in $T \setminus W$ (the subtree of T induced by $V \setminus W$).

Comment: This procedure is only called for some sets U whose size is bounded by a constant.

$n = |V|$
if $n = 1$ **then** $\mathbf{f}(\emptyset) = \mathbf{f}(U) = 1$ // In this case U is either $\emptyset$ or V.
Return $\mathbf{f}$
else if $n = 2$ **then** $\mathbf{f}(\emptyset) = 1 + x$
for all non-empty sets W **do** $\mathbf{f}(W) = 1$
Return $\mathbf{f}$
$v =$ Select-Root(T, U)
Consider v to be the root of T, and let d be the degree of v.
Let $v_1, \ldots, v_d$ be the neighbors of the root v.
For $i = 1, \ldots, d$, let $T_i = (V_i, E_i)$ be the subgraph of T induced by all the vertices reachable from v_i without going through v as an intermediate vertex.
// Thus the sets E_i form a partition of E, the sets $V_i \setminus \{v\}$ form a partition
// of $V \setminus \{v\}$, and $v \in V_i$ for all i.
$U = U \cup \{v\}$
for $i = 1$ **to** d **do**
$\mathbf{f}_i =$ Restricted-Matchings$(T_i, V_i \cap U)$
$S = \{T_1, \ldots, T_d\}$
while $|S| > 1$ **do**
Let T_i and T_j be two approximately smallest trees in S of sizes n_i and n_j respectively.
// Replace T_i and T_j by their union. Call it T_k.
$n_k = n_i + n_j - 1$
$S = S \setminus \{T_i, T_j\} \cup \{T_k\}$
for all $W \subseteq U$ **do**
if $v \in W$ **then**
$\mathbf{f}_k(W) = \mathbf{f}_i(W)\mathbf{f}_j(W)$
else
$\mathbf{f}_k(W) = \mathbf{f}_i(W)\mathbf{f}_j(W) - (\mathbf{f}_i(W) - \mathbf{f}_i(W \cup \{v\}))(\mathbf{f}_j(W) - \mathbf{f}_j(W \cup \{v\}))$
Now S is a singleton $\{T_k\}$ with $T_k = T$.
Return $\mathbf{f}_k$

Fig. 3. The procedure Restricted-Matchings

The algorithms are natural, easy to understand and yet efficient. Their correctness immediately follows from the following principles.

- The well known relationship between numbers of matchings and the coefficients of the characteristic polynomial of a tree.
- The characteristic polynomial $\chi(G;\lambda)$ of a union $G = (V_1 \cup V_2, E_1 \cup E_2)$ of disjoint graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ (with $V_1 \cap V_2 = \emptyset$) is the product of the characteristic polynomials $\chi(G_1;\lambda)$ and $\chi(G_2;\lambda)$. This is seen immediately from the block structure of the adjacency matrix in the definition of $\chi(G;\lambda)$ as a determinant.
- Under the same conditions the matching generating polynomials are multiplicative too. $f_M(G;x) = f_M(G_1;x)\, f_M(G_2;x)$. This follows from the fact that each matching in G_1 can be combined with each matching in G_2.
- For every vertex v with the set of neighbors $\{v_1, \ldots, v_k\}$ in any graph G, the number $\mathrm{match}(G, r)$ of r-matchings in G is decomposed as follows.

$$\mathrm{match}(G, r) = \mathrm{match}(G \setminus \{v\}, r) + \sum_{r=1}^{k} \mathrm{match}(G \setminus \{v, v_r\}, r-1)$$

- In a tree, every internal vertex v is an articulation point, meaning that almost all these graphs obtained by deleting vertices, decompose into connected components for which the product rule holds.

All these simple properties could be used to design a straightforward algorithm computing the matching generating polynomial by a simple tree traversal, computing the polynomial for the tree rooted at v recursively from the polynomials of the subtrees rooted at the children of v. The problem is that this natural algorithm runs in quadratic time.

On the positive side, this design immediately results in a linear time algorithm to count the number of maximal independent sets and maximum independent sets.

When computing the whole matching generating polynomial, we overcome the quadratic time problem by a cleverer selection of the articulation points v. The simple idea of splitting as evenly as possible is not enough. We also have to deal with the vertices of U. We don't solve just one matching problem, but one for each possible restriction on the vertices of U. We would also like to split the set U evenly. The time analysis shows that we can just switch back and forth as needed between the two objectives of splitting V and splitting U nicely.

In both cases, we want to select a point v that according to the current criterion is located in the center of the graph.

Definition 3. *For $T = (V, E)$, $U \subseteq V$, and $|U| \geq 2$, let* $\mathrm{Center}(T, U)$ *be one of the nodes $v \in V$ such that every tree in $T \setminus \{v\}$ (the subgraph of T induced by $V \setminus \{v\}$) contains at most $|U|/2$ points of U.*

For $U = V$, there are either one or two vertices v with this property. In the latter case, the procedure $\mathrm{Center}(T, U)$ in Figure 4 picks an arbitrary one of them. For

Procedure Select-Root:

Input: A tree $T = (V, E)$ and a subset $U \subseteq V$.
Output: A vertex v of T which will be viewed as the root of T.
Comment: Let $n_0 \geq 3$ be an integer constant. n_0 is an upper bound on the size of U. The choice of n_0 only affects the running time by a constant factor. $n_0 = 5$ might be the optimal choice.

if $|U| \geq n_0$ **then**
  Return Center(T, U)
else
  Return Center(T, V)

Fig. 4. The procedure Select-Root

$|E| \geq 2$ and U a set of leaves, the set of vertices with this property consists of the vertices of a path. If this path has positive length, then the procedure Center(T, U) picks any vertex of this path. A simple traversal of the tree T (with counting the number of vertices of U in the subtree of v on post-visiting v) finds a center in linear time.

An alternative approach is to keep $|U| \leq 2$. As before, for $|U| \leq 1$ Center(T, V) is called to pick a vertex minimizing the size of the largest tree in $V \setminus \{v\}$. Otherwise for $U = \{u_1, u_2\}$, Center(T, U) picks a vertex v on the path from u_1 to u_2, still minimizing the size of the largest tree in $V \setminus \{v\}$. This approach seems somewhat more efficient (by a constant factor), but is not analyzed here.

3 Time Complexity

As the previously cited papers, this paper uses the customary algebraic computation model. All arithmetic operations, including multiplications are counted as one step. This is not a serious problem, as all our numbers have at most a linear length in binary.

Analysing the procedure Restricted-Matchings, we first note that the size of U is under control as long as it is initially bounded by n_0. In the recursive call for T_i, the set $U_i = V_i \cap U \cup \{v\}$ plays the role of U.

Lemma 1. *Let $n_0 \geq 3$ be the constant used in the procedure Select-Root. If the procedure Restricted-Matchings is called with $|U| \leq n_0$ then all the recursive calls are with $|U_i| \leq n_0$. Furthermore, if $|U| = n_0$, then all $|U_i| < n_0$.*

Proof. For $|U| < n_0$ the set U_i satisfies $|U_i| \leq |U| + 1$, while for $|U| = n_0$ the algorithm is designed to split U evenly, resulting in the inequality $|U_i| \leq \lfloor |U|/2 \rfloor + 1 \leq \lfloor n_0/2 \rfloor + 1 < n_0$ for $n_0 \geq 3$. □

Therefore, as $U = \emptyset$ at the beginning, the size of the set U will stay bounded by $n_0 \geq 3$, and U does not even reach the bound n_0 twice in a row.

Let $m = n - 1$ be the number of edges in T. Assume $m \geq 1$, as the one vertex case is trivial and does not show up during recursive calls.

Lemma 2. *For $m \geq 1$ and suitable constants c, c', and c'', the running time of the procedure Restricted-Matchings is at most $c\,m \lg^2 m + c''m$ for $|U| < n_0$ and at most $c\,m \lg^2 m + c'\,m \lg m + c''m$ for $|U| = n_0$.*

Proof. The lemma trivially holds for $m = 1$. Let $m \geq 2$ and assume the lemma is true for all trees with less than m edges.

Recall that the procedure Restricted-Matchings partitions the tree T edge-wise into trees $T_1, \ldots, T_d$ with $T_i = (V_i, E_i)$, $|E_i| = m_i$, and for $|U| < n_0$, the sizes m_i are bounded by $m/2$ for all i. After the recursive calls for these trees T_i with common root v, repeatedly pairs of approximately smallest trees are merged into single trees until there is just one tree left, i.e., T has been reassembled. Obviously, there is at most one tree T' among the trees T_i with $|U_i| = n_0$. Let m' be the number of edges of T'. Let $m_{\min} = \min_i m_i$.

Claim: If after some sequence of merges of pairs of trees, we have the trees $T_1, \ldots, T_{d'}$, then the time spent for the recursive calls and the merges together has been at most

$$t(d') = c \sum_{i=1}^{d'} m_i \lg^2 m_i + c'\, m_{\min} \lg m_{\min} + b\, m' \lg m' + c''m \tag{1}$$

where c, c' and c'' are from the lemma and b is defined by $b = c'$ if the tree T' (with $|U_i| = n_0$) exists and $b = 0$ otherwise.

For the total time, until all merges have been done, i.e., for $d' = 1$, we will show a different bound t' later.

The proof of the claim is by induction on the number of merges. The base case (just before any merges) follows immediately from the inductive hypothesis of the lemma, without any need for the second term $c'\, m_{\min} \lg m_{\min}$. For the inductive step, we look at the difference $t(d') - t(d' + 1)$ of the allowed time after and before the merge of two trees T_i and T_j into T_k. We show in each case that this time difference is enough to perform the merge, i.e., to compute the polynomial for T_k from the polynomials for T_i and T_j.

First note that none of the four terms in $t(d')$ decreases during a merge (i.e., as d' decreases by 1). The first term always increases by

$$c\,(m_i + m_j) \lg^2 (m_i + m_j) - c\, m_i \lg^2 m_i - c\, m_j \lg^2 m_j > 0$$

The second term increases when $m_{\min}$ increases. The last two terms clearly don't decrease.

We consider two kind of merges depending on whether the merged trees are of similar size or not. W.l.o.g., we assume $m_i \leq m_j$.

Case "not similar": Assume T_i and T_j are merged with $1 \leq m_{\min} = m_i < m_j/4$, and $\{T_i, T_j\}$ are approximately minimal, i.e., $m_\ell > m_j/2$ for all $\ell \neq i$. Here we do not assume $m_j \leq m/2$. For $|U| = n_0$, it is possible to have a large tree with m_j very close to m. Now the second term in $t(d') - t(d' + 1)$ increases by at least

$$\begin{aligned}
& c' \frac{m_j}{2} \lg \frac{m_j}{2} - c' m_{\min} \lg m_{\min} \\
& > c' \frac{m_j}{2} \lg \frac{m_j}{2} - c' \frac{m_j}{4} \lg \frac{m_j}{4} \\
& > c' \frac{m_j}{4} \lg \frac{m_j}{2} \\
& > c' \frac{m_j}{8} \lg m_j \quad (\text{as } m_j > 4) \\
& > C' m_j \lg m_j
\end{aligned}$$

First C' is chosen large enough to make it possible to do the last merge in time $C' m_j \lg m_j$, i.e., to do the multiplications of $O(1)$ pairs of polynomials of degree m_i and m_j respectively using the fast Fourier transformation (FFT). Then we make sure c' is chosen sufficiently large that the last inequality holds.

Case "similar": Assume T_i and T_j are merged with $m_i \le m_j \le 4m_i$. Now the first term in $t(d') - t(d'+1)$ increases by

$$\begin{aligned}
& c\,(m_i + m_j) \lg^2 (m_i + m_j) - c\, m_i \lg^2 m_i - c\, m_j \lg^2 m_j \\
& \ge c\,(m_i + m_j) \lg^2 (\tfrac{5}{4} m_j) - c\, m_i \lg^2 m_j - c\, m_j \lg^2 m_j \\
& = c\,(m_i + m_j)((\lg \tfrac{5}{4} + \lg m_j)^2 - \lg^2 m_j) \\
& = c\,(m_i + m_j)(2 \lg \tfrac{5}{4} \lg m_j + \lg^2 \tfrac{5}{4}) \\
& > C\, m_j \lg m_j
\end{aligned}$$

Again C is first chosen large enough to make it possible to do the last merge in time $C\, m_j \lg m_j$, i.e., to do the multiplications of $O(1)$ pairs of polynomials of degree m_i and m_j respectively using FFT. Then we make sure c is chosen sufficiently large that the last inequality holds. This proves the claim.

At this point, we should notice that Claim (1) is not always strong enough to show the inductive step in the induction proof of the lemma. Indeed we can do better during the last merge. We claim a different bound t' instead of just $t(1)$ to hold after the last merge, when we have just one tree $T_k = T$.

$$t' = c\, m \lg^2 m + a\, m \lg m + c'' m \tag{2}$$

where $a = c'$ if $|U| = n_0$ (where U is the set associated with the tree T), and $a = 0$ otherwise.

The case with $|U| = n_0$ and therefore $|U_i| < n_0$ for all i causes no problem. Then $b = 0$, $m_{\min} = m$, and the first time bound (1) implies the second (2).

In the case $|U| < n_0$, the last merge has to be handled separately. We show that this merge is always balanced and therefore significantly cheaper. We are left with two trees with m_i and m_j edges to be merged into a tree of $m = m_k = m_i + m_j$ edges. We assume $m_i \le m_j$. We claim $m_j < \frac{4}{5} m$. Otherwise, the large tree with more than $\frac{4}{5} m$ edges would have been produced by a merge involving a tree of size at least $\frac{2}{5} m$, omitting a tree of size at most $\frac{1}{5} m$, contradicting the rule of always merging approximately smallest trees.

Now the difference of bounds is

$$\begin{aligned} t' - t(2) &= c\,m \lg^2 m - c(m_i \lg^2 m_i + m_j \lg^2 m_j) - c'\, m_{\min} \lg m_{\min} - b\, m' \lg m' \\ &> c\, m_i(\lg^2 m - \lg^2 m_i) + c\, m_j(\lg^2 m - \lg^2 m_j) - 2c' m \lg m \\ &\geq c\, m(\lg^2 m - \lg^2 m_j) - 2c' m \lg m \\ &> c\, m(\lg m + \lg m_j)(\lg m - \lg m_j) - 2c' m \lg m \\ &= c\, m \lg(m m_i) \lg(m/m_j) - 2c' m \lg m \\ &> c\, m \lg m \lg \tfrac{5}{4} - 2c' m \lg m \\ &> C\, m \lg m \end{aligned}$$

Once more, C has been chosen large enough to make it possible to do the last merge in time $C\, m \lg m$, i.e., to do the multiplications of $O(1)$ pairs of polynomials of degree m_i and m_j respectively. Then we make sure c is chosen sufficiently large that the last inequality holds. □

Lemma 2 immediately implies the desired complexity result for the procedure Restricted-Matchings and therefore also for the algorithm Matchings.

Theorem 1. *For $|U| \leq n_0$, the running time of the algorithm Matchings is $O(n \log^2 n)$.*

4 Other Problems

The algorithms and their analysis easily transfer to many other counting problems, like computing the independent set generating polynomial, the vertex cover generating polynomial and so on. Another example is computing the number of 3-colorings which color exactly r vertices being colored red, simultaneously for every r. All these problems can be solved in time $O(n \log^2 n)$ for trees with basically the same algorithm. All we need is that these are all local properties. If a set is not independent, then you can put your finger on an edge where both incident vertices are selected.

Indeed the algorithms for these problems are even slightly easier than computing the matching generating polynomial, because the counted objects are sets of vertices not sets of edges. Instead of $\mathbf{f}(W)$ whose r-th coefficient is the number of matchings not involving the vertices of W, we would use $\mathbf{f}(U, W)$ whose r-th coefficient is the number of independent sets including all the vertices of $(U \setminus W) \cap V$ and excluding all the vertices of $W \cap V$, when counting independent sets. This would change the "if" clause of the procedure Restricted-Matchings near the end of the procedure to

$$\mathbf{f}_k(U, W) = \mathbf{f}_i(U, W)\, \mathbf{f}_j(U, W)$$

and the corresponding "else" clause to

$$\mathbf{f}_k(U, W) = \mathbf{f}_i(U, W)\, \mathbf{f}_j(U, W)/x$$

Note that now in both cases, the number of independent sets is just the product of the the number of independent sets in the two subtrees. In the second case, the vertex v is double counted, as it is in the independent sets of both subtrees. This is corrected by the division by x.

This change would be exactly the same, if we wanted to count other locally testable sets of vertices, like the numbers of vertex covers. But there are additional changes of the initialization. These changes are different for different kinds of polynomials. It would be a bit tedious to describe the complete initialization, because there are so many cases. For example, for $n = 2$ with one vertex $u \in U \setminus W$ (i.e., u is required in the set), and the other vertex $v \in V \setminus U$ (i.e., v is allowed by not required in the set), $\mathbf{f}(U, W) = x + x^2$ for Vertex Cover, but $\mathbf{f}(U, W) = x$ for Independent Set, as in both cases $\{u\}$ is the only allowed singleton set, while $\{u, v\}$ is a vertex cover but not an independent set.

5 Graphs of Bounded Tree-Width

Things get much more tedious, but the method clearly carries through graphs of bounded tree-width. Instead of the recursive calls having to deal with all ways of handling one additional vertex v of the tree, now recursive calls have the additional task of dealing with all possible ways of handling all the graph vertices assigned to the same tree vertex v. Naturally, the running time is exponential in the tree-width, but that is still just a constant. The dependence on n remains $O(n \log^2 n)$.

There is an extensive literature on graph polynomials for graphs of bounded tree-width. A main focus is on parametrized complexity of counting and evaluation problems on graphs definable in Monadic Second Order Logic [10,11,12]. For bounded tree-width these problems are solvable in polynomial time. The resulting running time is $O(f(k)n^4)$ with the dependence on the parameter $f(k)$ double exponential. When our algorithm is applied to graph polynomials for graphs of bounded tree width, then $f(k)$ is singly exponential.

There is some confusion caused by a linear time algorithm for the interlace polynomial for graphs of bounded tree-width [13]. This is a more complicated multivariate polynomial. It is important to notice that these authors have a different notion of computing a polynomial. While they compute one value of a polynomial in linear time, we compute all the coefficients of our polynomials in almost linear time.

6 Final Remark

We have presented a simple algorithmic paradigm with very wide applicability for counting problems in graphs of bounded tree-width. It always saves a factor of order $n/\log^2 n$ over previously known methods. A detailed description in general terms will be quite tedious, even though, there is no principle hurdle.

We have decided to present the method with detailed descriptions and proofs for the special, but interesting case of computing the characteristic polynomial

of a tree. This problem has been investigated before, and it is not astonishing that previous progress has stopped at the "natural" bound of $O(n^2)$.

Only a new multitasking divide and conquer method has allowed to obtain a significantly more efficient algorithm. The method allows to divide according to two different natural strategies, taking turns when needed, and basically reaching both goals.

References

1. Biggs, N.: Algebraic graph theory, 2nd edn. Cambridge Mathematical Library. Cambridge University Press, Cambridge (1993)
2. Tinhofer, G., Schreck, H.: Computing the characteristic polynomial of a tree. Computing 35(2), 113–125 (1985)
3. Keller-Gehrig, W.: Fast algorithms for the characteristic polynomial. Theor. Comput. Sci. 36(2,3), 309–317 (1985)
4. Bürgisser, P., Clausen, M., Shokrollahi, M.A.: Algebraic complexity theory. Grundlehren der Mathematischen Wissenschaften or Fundamental Principles of Mathematical Sciences, vol. 315. Springer, Berlin (1997); With the collaboration of Thomas Lickteig
5. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. Journal of Symbolic Computation 9(3), 251–280 (1990)
6. Fricke, G.H., Hedetniemi, S., Jacobs, D.P., Trevisan, V.: Reducing the adjacency matrix of a tree. Electron. J. Linear Algebra 1, 34–43 (1996) (electronic)
7. Mohar, B.: Computing the characteristic polynomial of a tree. J. Math. Chem. 3(4), 403–406 (1989)
8. Jacobs, D.P., Machado, C.M.S., Trevison, V.: An $O(n^2)$ algorithm for the characteristic polynomial of a tree. J. Combin. Math. Combin. Comput. 54, 213–221 (2005)
9. Ellis-Monaghan, J., Merino, C.: Graph polynomials and their applications ii: Interrelations and interpretations (2008)
10. Courcelle, B., Makowsky, J.A., Rotics, U.: On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. Discrete Appl. Math. 108(1-2), 23–52 (2001)
11. Makowsky, J., Marino, J.: Farrell polynomials on graphs of bounded tree width. Advances in Applied Mathematics 30, 160–176 (2003)
12. Makowsky, J.A.: From a zoo to a zoology: Descriptive complexity for graph polynomials. In: Beckmann, A., Berger, U., Löwe, B., Tucker, J.V. (eds.) CiE 2006. LNCS, vol. 3988, pp. 330–341. Springer, Heidelberg (2006)
13. Bläser, M., Hoffmann, C.: Fast computation of interlace polynomials on graphs of bounded treewidth. CoRR abs/0902.1693 (2009); 35 pages informal publication