Computation in permutation groups: counting and randomly sampling orbits

Leslie Ann Goldberg

Abstract

Let Ω be a finite set and let G be a permutation group acting on Ω. The permutation group G partitions Ω into orbits. This survey focuses on three related computational problems, each of which is defined with respect to a particular input set $\mathcal{I}$. The problems, given an input $(\Omega, G) \in \mathcal{I}$, are (1) count the orbits (exactly), (2) approximately count the orbits, and (3) choose an orbit uniformly at random. The goal is to quantify the computational difficulty of the problems. In particular, we would like to know for which input sets $\mathcal{I}$ the problems are tractable.

1 Introduction

Let Ω be a finite set and let G be a permutation group acting on Ω. The permutation group G partitions Ω into *orbits*: Two elements of Ω are in the same orbit if and only if there is a permutation in G which maps one element to the other. This survey focuses on three related computational problems, each of which is defined with respect to a particular input set $\mathcal{I}$:

1. Given an input $(\Omega, G) \in \mathcal{I}$, count the orbits.

2. Given an input $(\Omega, G) \in \mathcal{I}$, approximately count the orbits.

3. Given an input $(\Omega, G) \in \mathcal{I}$, choose an orbit uniformly at random.

The goal is to quantify the computational difficulty of the problems. In particular, we would like to know for which input sets $\mathcal{I}$ the problems are tractable.

Many interesting orbit-counting problems come from the setting of "Pólya theory". In this setting, Σ is a fixed alphabet of size at least two. For every (infinite) set $\mathcal{G}$ of permutation groups, we get an input set $\mathcal{I}(\mathcal{G})$. In particular, the group $G \in \mathcal{G}$ corresponds to the input $(\Sigma^m, \widehat{G}) \in \mathcal{I}(\mathcal{G})$, where m is the degree of G, Σ^m is the set of length-m words over alphabet Σ, and $\widehat{G}$ is a permutation group acting on Σ^m which is induced by G. We will later give a precise definition of $\widehat{G}$ (as a function of G), but the rough idea is as follows: Every permutation $g \in G$ induces exactly one permutation $\hat{g} \in \widehat{G}$. The image of a word $\alpha \in \Sigma^m$ under $\hat{g}$ is the word β that is obtained from α by permuting the m alphabet symbols in α (applying g to the "positions" of the symbols). We use the notation $\mathcal{P}$ to denote the set of *all* permutation groups, so $\mathcal{I}(\mathcal{P})$ is the input set corresponding to all inputs in the Pólya-theory framework (over a fixed alphabet Σ).

Problems 1–3 are studied in Sections 3, 4 and 5 respectively. In those sections, we will give the precise complexity-theory definitions which we will need to formally capture the notion of "tractability". We will also give appropriate references, tracing the development of ideas. In this section, we introduce the area by giving a very high-level sketch, stating (roughly) what is known about each of the three problems, without giving details, definitions, or references.

There are natural input sets for which exact orbit-counting is tractable. Here is an example. Suppose that Ω_n is the set of all n-vertex trees and that G_n is the permutation group acting on Ω_n which is induced by vertex permutations. The orbits of Ω_n under G_n correspond to isomorphism classes. That is, there is one orbit for each *unlabelled* n-vertex tree. Unlabelled trees can be counted quickly using classical methods based on generating functions and dynamic programming. Thus, the orbit-counting problem is tractable for the input set $\mathcal{I} = \{(\Omega_n, G_n)\}$.

Perhaps surprisingly, the orbit-counting problem becomes intractable if we make the input set slightly more complicated: Suppose that each input corresponds to a tree T, and orbits correspond to unlabelled subtrees of T. It turns out that this orbit-counting problem is #P-complete. We will discuss the complexity class #P in Section 3. To give a rough idea, counting problems which are complete in #P are equivalent in difficulty to counting the number of satisfying assignments of a Boolean formula, and this is believed to be very demanding computationally. Thus, it is unlikely that there is a polynomial-time algorithm for counting unlabelled subtrees of a tree T. What happens when we make the input set still more complicated? Suppose now that each input corresponds to a graph H, and orbits correspond to unlabelled subtrees of H. By the previous result, the new problem (counting unlabelled subtrees of a graph) is at least as difficult as #P. But perhaps it is more difficult? Whether or not it is more difficult is unknown, but a complexity-theoretic result of Toda implies that it cannot be much more difficult. The new problem is complete in the complexity class $\mathrm{FP}^{\#\mathrm{P}}$, which, as we will see later, is not so different from #P. The problem of counting orbits in the Pólya-theory setting (with input set $\mathcal{I}(\mathcal{P})$) is also complete in $\mathrm{FP}^{\#\mathrm{P}}$. There are several natural problems for which the complexity of orbit-counting is unresolved. For example, there is no known efficient algorithm for exactly counting unlabelled n-vertex graphs. It seems plausible that this problem is computationally difficult, but we seem to lack the complexity-theory machinery to quantify the difficulty of such problems. These problems are members of the complexity class $\#\mathrm{P}_1$, which will be discussed in Section 3.

There are many known examples of counting problems which are #P-complete to solve exactly but for which good (efficient) approximation algorithms exist. Interestingly, we do not know any natural[1] orbit-counting

[1] Technically, every counting problem without orbits can be expressed as an orbit-counting problem in which the relevant permutation group contains only the identity, but this is not what we mean by a "natural" orbit-counting problem.

problems which have this property, though perhaps this is just because orbit-counting problems have not been sufficiently studied.

There are examples of problems for which the complexity of exact orbit-counting is unresolved, but approximate orbit-counting is known to be tractable. For example, there is an efficient algorithm for approximately counting unlabelled n-vertex graphs. It is worth pointing out we do not know any efficient algorithms for this problem using traditional methods such as generating functions and dynamic programming. The only algorithm which is known relies upon a reduction from almost-uniform sampling, which we will discuss presently (see also Section 5.6).

As we mentioned before, some #P-complete counting problems have efficient approximation algorithms. For other #P-complete counting problems, the corresponding approximation problem can be shown to be "complete" in a formal sense which implies that there is no efficient approximation algorithm under standard complexity-theoretic assumptions. It would be very interesting to know whether the approximate orbit-counting problem is complete in this sense for the set of Pólya-theory inputs $\mathcal{I}(\mathcal{P})$. This seems to be a very difficult question. At present, the most that can be said is that the related question of approximately counting orbits over a coset (rather than over a group) is complete.

For many computational counting problems, there is a close connection between the complexity of approximately counting and the complexity of sampling uniformly at random. However, with a few exceptions, which we will discuss, these connections break down for orbit-counting problems. Thus, sampling must be studied separately from approximate counting.

Most of the work on sampling orbits has focused on three methods: inductive sampling, the orbit-sampling process, and rejection sampling. "Inductive sampling" can be used when generating functions for enumerating orbits can be efficiently evaluated. The idea is to find a recurrence for the coefficients of the generating function, which can be computed by dynamic programming. The recurrence should show how to express outputs corresponding to a given instance of the problem in terms of outputs to smaller instances. Sampling is then done recursively: The coefficients are used to probabilistically select the appropriate (recursive) sub-problem. For example, unlabelled n-vertex trees can be sampled in this way.

The "orbit-sampling process" is inspired by the orbit-counting lemma of Cauchy, Frobenius and Burnside. This lemma says that each orbit is represented $|G|$ times in the set $\Upsilon(\Omega, G)$, which is the set of pairs (α, g) such that $\alpha \in \Omega$, $g \in G$ and g fixes α (that is, g maps α to itself). The orbit-sampling process is a Markov chain with state space Ω. To make a transition from the state α, the chain first picks uniformly-at-random a permutation $g \in G$ which fixes α and then chooses the new state uniformly-at-random from the subset of Ω which is fixed by g. The orbit-sampling lemma can be used to show that the stationary distribution of this process is uniform on orbits. If the input set is chosen such that (1) each transition can be implemented in polynomial time, and (2) the Markov chain is rapidly mixing, then the orbit-process gives an efficient sampler for orbits.

Not too much is known about the mixing-time of the orbit-sampling process, even in the Pólya-theory setting. It is not rapidly-mixing for the complete set of inputs $\mathcal{I}(\mathcal{P})$. There is also an infinitely-large subset of $\mathcal{I}(\mathcal{P})$ for which the transitions can be implemented in polynomial time, but the process is still not rapidly mixing. The process is known to be rapidly mixing if the set $\mathcal{G}$ of permutation groups is either (1) the set of all symmetric groups S_n, or (2) the set of all cyclic groups (all permutation groups with a single generator), but nothing else is known. In particular, it is not even known whether the orbit-sampling process is rapidly-mixing for the situation in which orbits correspond to unlabelled n-vertex graphs.

As we mentioned earlier, there is an alternative efficient algorithm for sampling unlabelled n-vertex graphs. This algorithm uses the "rejection sampling" method, which will be explained in Section 5.6. The algorithm can be extended to the general orbit-sampling framework. Whether or not it leads to an efficient sampling algorithm depends upon the input set $\mathcal{I}$. What is clear is that it will not work unless the identity permutation accounts for a sufficiently large fraction of the pairs in $\Upsilon(\Omega, G)$. That is, a typical member of Ω must not be fixed by too many permutations in G.

An example of an input set which does not have this property is as follows: Each input corresponds to a degree-sequence in which every degree is bounded. Orbits correspond to unlabelled connected multigraphs with the given degree sequence. Here the degree sequence may contain many vertices of degree 1 or 2, in which case a typical member of Ω is a multigraph which is fixed by many automorphisms. The ideas which have been described so far can be combined to give an efficient sampling algorithm for this problem. Nevertheless, we lack good general techniques for orbit-sampling, especially when the objects in Ω have many symmetries.

The final section of the survey is devoted to a problem which is related to that of sampling and counting orbits – namely, the problem of listing orbits. Not too much is known about the problem, and the section gives pointers to some recent work in the area.

2 Definitions and Preliminaries

Let Ω be a finite set and let G be a permutation group acting on Ω. If $\alpha \in \Omega$ and $g \in G$, we write α^g to denote the image of α under g. We write G_α to denote the subgroup of G consisting of the permutations in $\{g \in G \mid \alpha^g = \alpha\}$. We define the relation $\sim$ on Ω in which $\alpha \sim \beta$ if and only if there is a permutation $g \in G$ such that $\alpha^g = \beta$. The relation $\sim$ partitions Ω into equivalence classes, which are called *orbits*. We use the notation α^G to denote the orbit $\{\alpha^g : g \in G\}$ containing α and the notation $\Phi(\Omega, G)$ to denote the set of orbits. For each permutation $g \in G$, we let $\mathrm{fix}(g)$ denote the set $\{\alpha \in \Omega \mid \alpha^g = \alpha\}$. We let $\Upsilon(\Omega, G)$ denote the set

$$\Upsilon(\Omega, G) = \{(\alpha, g) \mid \ \alpha \in \Omega \text{ and } g \in G \text{ and } \alpha \in \mathrm{fix}(g)\ \}.$$

The following lemma was known to Cauchy and Frobenius (see [40]) but is often called "Burnside's Lemma". Following Cameron [7], we call it the "orbit-counting" lemma.

Lemma 2.1 The orbit-counting lemma *Let G be a permutation group on the finite set Ω. Then for each orbit $\Delta \in \Phi(\Omega, G)$ we have*

$$|\{(\alpha, g) \in \Upsilon(\Omega, G) \mid \alpha \in \Delta\}| = |G|,$$

so

$$|\Upsilon(\Omega, G)| = |\Phi(\Omega, G)|\,|G|.$$

Example 2.2 *Let Ω_n be the set of all n-vertex graphs and let G_n be the permutation group acting on Ω_n which is induced by vertex permutations. That is, G_n has $n!$ permutations — one for each permutation of the n vertices. If π is a permutation of the vertices and α is a graph in Ω_n then the image of α under the permutation corresponding to π is the graph obtained from α by applying π to the vertices. The orbits of Ω_n under G_n correspond to isomorphism classes. Thus, we can think of the orbits as representing the set of* unlabelled *n-vertex graphs.*

We will let $\mathcal{U}_{\Omega,G}$ denote the uniform distribution on orbits. That is, for each orbit Δ in $\Phi(\Omega, G)$, the probability of Δ in $\mathcal{U}_{\Omega,G}$ (denoted $\mathcal{U}_{\Omega,G}(\Delta)$) is $1/|\Phi(\Omega, G)|$.

We will measure the distance between two probability distributions D_1 and D_2 over the discrete sample space Ψ using the *total variation distance* metric. Namely,

$$\mathrm{d_{tv}}(D_1, D_2) = \max_{A \subseteq \Psi} |D_1(A) - D_2(A)| = \frac{1}{2} \sum_{x \in \Psi} |D_1(x) - D_2(x)|.$$

2.1 A special case of Pólya's theorem

Many of the orbit-counting problems which we will consider come from the setting of Pólya theory. We will not be using the fully general version of Pólya's theorem. Instead, we will restrict our attention to the special case of the theorem that we define in this section. Suppose that $\Sigma = \{0, \ldots, k-1\}$ is a finite alphabet and that G is a group of permutations of the set $\{0, \ldots, m-1\}$, which we denote $[m]$. For every permutation $g \in G$, let $c(g)$ denote the number of cycles in g. Let Ω be the set Σ^m of length-m words over alphabet Σ. The group G has a natural action on Ω which is induced by permuting the "positions" $0, \ldots, m-1$ of the alphabet symbols in the words. In particular, if $\alpha = a_0 a_1 \ldots a_{m-1}$ is a word in Ω then the image of α under the induced action of g is the word $\beta = b_0 b_1 \ldots b_{m-1}$, in which, for all $j \in [m]$, b_j is $a_{j^{g^{-1}}}$ That is, b_j is the element a_i such that $i^g = j$. To avoid confusion, we use the symbol $\widehat{G}$ to denote the permutation group on Ω which is induced by G and we use the symbol $\hat{g}$ to denote the permutation of Ω which is induced by permutation $g \in G$.

Now $\mathrm{fix}(\hat{g})$ has $k^{c(g)}$ elements. In particular, if a word α is in $\mathrm{fix}(\hat{g})$, then all of the positions which form a single cycle of g must have the same alphabet symbol in α. There are k possible symbols which can be chosen. Thus the orbit-counting lemma (Lemma 2.1) gives us the following special case of Pólya's theorem.

Lemma 2.3 Pólya's theorem *If $\Sigma = [k]$ is a finite alphabet, $\Omega = \Sigma^m$, and G is a permutation group on $[m]$, then*

$$|\Phi(\Omega, \widehat{G})| = \frac{1}{|G|} \sum_{g \in G} k^{c(g)}.$$

Example 2.4 *If $\Sigma = \{0, 1\}$ and G is the symmetric group on $[m]$ then the $m+1$ orbits consist of, for each $i \in [m+1]$, those words in Σ^m with the symbol "1" in i positions.*

Example 2.5 *The set of unlabelled n-vertex graphs (Example 2.2) can be encoded as orbits in the Pólya-theory setting by using words in Σ^m to encode graphs. In particular, the graph H can be represented by the word corresponding to the upper-diagonal part of H's adjacency matrix.*

Further examples can be found in surveys such as [8] and [45].

2.2 Computational questions

We work in the following computational framework which is similar to that of [25]. We specify the input set $\mathcal{I}$, where each input in $\mathcal{I}$ consists of a set Ω and a permutation group G. The inputs are represented in a concise manner, which depends upon $\mathcal{I}$. We study the following computational problems.

1. *exact counting:* Given an input $(\Omega, G) \in \mathcal{I}$, output $|\Phi(\Omega, G)|$.

2. *approximate counting:* Given an input $(\Omega, G) \in \mathcal{I}$ and an accuracy parameter $\epsilon \in (0, 1)$, output an integer random variable Y satisfying

$$\Pr\left(e^{-\epsilon} \le \frac{Y}{|\Phi(\Omega, G)|} \le e^{\epsilon}\right) \ge \frac{3}{4}.$$

3. *almost-uniform sampling:* Given an input $(\Omega, G) \in \mathcal{I}$, and an accuracy parameter $\epsilon \in (0,1]$, output a random variable α. Typically, α will be a member of Ω, and will be viewed as a representative of its orbit, but it will be technically useful to allow sampling algorithms to sometimes produce other outputs.

   We measure the accuracy of a sampling algorithm by constructing a distribution D based on the output distribution of the algorithm. The domain of D is taken to be $\Phi(\Omega, G) \cup \{\perp\}$, where $\perp$ is an "error" symbol, which records the fact that the output does not represent an orbit. For each orbit $\Delta \in \Phi(\Omega, G)$, the probability of Δ in D is defined to be $\Pr(\alpha \in \Delta)$. Therefore, the probability of $\perp$ in D is equal to $1 - \sum_{\Delta \in \Phi(\Omega,G)} \Pr(\alpha \in \Delta)$. The algorithm is an *almost-uniform sampler* if and only if $\mathrm{d}_{\mathrm{tv}}(D, \mathcal{U}_{\Omega,G}) \leq \epsilon$.

For each *particular* input set $\mathcal{I}$, we get a particular exact counting problem, approximate counting problem, and almost-uniform sampling problem. We will usually discuss the representation of the inputs when the particular problem is discussed. Typically, we will represent inputs in a concise manner.

In the Pólya-theory setting, we will adopt the following notation from the introduction. The alphabet Σ is fixed. For any permutation group G, we will let $m(G)$ denote the degree of G. For any set $\mathcal{G}$ of permutation groups, $\mathcal{I}(\mathcal{G})$ denotes the input set corresponding to $\mathcal{G}$. In particular,

$$\mathcal{I}(\mathcal{G}) = \{(\Sigma^{m(G)}, \widehat{G}) \mid G \in \mathcal{G}\}.$$

We use the notation $\mathcal{P}$ to denote the set of *all* permutation groups, so $\mathcal{I}(\mathcal{P})$ is the input set corresponding to all inputs. The input $(\Sigma^m, \widehat{G})$ will be presented as a set of $O(m)$ generators for G.[2]

For convenience (in applying complexity-theoretic definitions), we will assume that all inputs to computational problems are encoded as words over the binary alphabet $\{0,1\}$. This typically does not present any problems. For example a generator of a degree-m permutation group can be encoded as a binary word of length $O(m \log m)$. Note that the input size is typically much smaller than the size of Ω or G. In the Pólya-theory setting, the size of Ω is k^m and the size of $\widehat{G}$ can be as large as $m!$, but the size of the input is bounded from above by a polynomial in m. We are interested in knowing for which input sets $\mathcal{I}$ the computational problems are tractable, in a sense which will be made clear as we go along.

[2] The construction of small generating sets is beyond the scope of this article, but Chapter 1 of [7] describes several such constructions, due to Schreier, Sims, Jerrum, McIver and Neumann.

3 Exact counting

In Sections 3.2 and 3.3 we will see that for many natural input sets $\mathcal{I}$ the exact orbit-counting problem is #P-hard. Thus, it is as difficult as counting the number of satisfying assignments of a Boolean formula. In order to give details, we need some definitions, which are given in Section 3.1.

3.1 The complexity class #P

Following Valiant [50], we say that a function $f : \{0,1\}^* \to \mathbb{N}$ is in the complexity class FP if it can be computed by a deterministic polynomial-time Turing machine. We say that it is in #P if there is a nondeterministic polynomial-time Turing machine M such that for all $x \in \{0,1\}^*$ the number of accepting computations of M on input x is $f(x)$. A *polynomial-time Turing reduction* from a function $f : \{0,1\}^* \to \mathbb{N}$ to a function $g : \{0,1\}^* \to \mathbb{N}$ is a deterministic polynomial-time *oracle* Turing machine which, whenever it is supplied with an "oracle" for g, can compute f. Thus, the reduction shows how to compute f in polynomial time, assuming that we have an imaginary means for computing g in polynomial time. A counting problem, i.e., a function $f : \{0,1\}^* \to \mathbb{N}$ is said to be #P-*hard* if every function in #P is polynomial-time Turing-reducible to f. If, in addition, $f \in$ #P, then it is said to be #P-*complete.* A #P-complete problem is equivalent in computational difficulty to problems such as counting the number of satisfying assignments of a Boolean formula, or evaluating the permanent of a 0,1-matrix, which are widely believed to be intractable. For background information on #P and its completeness class, see, for example, [14] or [44].

3.2 Automating Pólya theory

In this section, we see that the problem of counting orbits in the Pólya-theory setting is equivalent in computational difficulty to solving a #P-complete problem. Let $\Sigma = [k]$ be a fixed alphabet with $k > 1$. Consider the following computational problem.

Name. #PÓLYAORBITS.

Instance. $O(m)$ generators for a group G of permutations of $[m]$.

Output. $|G| \cdot |\Phi(\Sigma^m, \widehat{G})|$.

The size of a permutation group can be computed in polynomial time from an arbitrary set of generators (see [7]). Thus, #PÓLYAORBITS is computationally equivalent to the orbit-counting problem with input set $\mathcal{I}(\mathcal{P})$. The following theorem quantifies the computational difficulty of this problem.

Theorem 3.1 *[17] #PÓLYAORBITS is #P-complete.*

Pólya's Theorem (Lemma 2.3) tells us that the appropriate output of #PÓLYAORBITS is $\sum_{g \in G} k^{c(g)}$. From this, it is not difficult to show that #PÓLYAORBITS is in #P. Thus, to prove Theorem 3.1, we need only show that it is #P-hard. The #P-hardness follows from Theorem 2 of [17] and an alternative proof has been given by Jerrum in [27]. Nevertheless, in order to generalise the result in Section 4, we will need a hardness proof which does not use interpolation, so we provide such a proof here.

We start with some definitions. A "cut" of a graph is an unordered partition (S, T) of its vertex set. The "cut edges" corresponding to the cut are those edges of the graph which have one endpoint in S and the other in T. Since the partition (S, T) is unordered, a cut of a connected graph is uniquely determined by the set of cut edges. The "size" of the cut is the number of cut edges. Lemma 12 of Jerrum and Sinclair's paper [30] shows that the following problem is #P-complete.

Name. #LARGECUT.

Instance. A positive integer j and a connected non-bipartite graph H in which no cuts are larger than size j.

Output. The number of size-j cuts of H.

Thus, Theorem 3.1 follows from the following lemma.

Lemma 3.2 *There is a polynomial-time Turing-reduction from #LARGECUT to #PÓLYAORBITS.*

Proof Let j and H be an instance of #LARGECUT. Let V denote the vertex set of H and let E denote the edge set of H. Let $r = |V|^2$. Let G be the degree-$2r|E|$ permutation group constructed as follows. For each edge $e \in E$ and each $i \in [r]$, we will have an object $a_{e,i}$ and an object $b_{e,i}$. For each vertex $v \in V$ we will have a permutation g_v. The action of g_v on the set $\bigcup_{e \in E} \bigcup_{i \in [r]} \{a_{e,i}, b_{e,i}\}$ is as follows. For every edge $e \in E$ which is incident on v, and for every $i \in [r]$, g_v transposes $a_{e,i}$ and $b_{e,i}$. Let G be the group generated by the permutations $\bigcup_v g_v$.

Since the permutations in $\{g_v\}$ commute and have order 2, the permutations in G correspond to subsets of V. The points $a_{e,1}$ and $b_{e,1}$ are transposed in a given permutation if and only if exactly one of the endpoints of e is in the corresponding subset of V. Thus, the permutations in G are in one-to-one correspondence with the cuts of H. A cut of size ℓ corresponds to a permutation with $2|E|r - \ell r$ cycles.

Now let h be the permutation which transposes every pair $(a_{e,i}, b_{e,i})$. Since H is not bipartite, $h \notin G$. Let G' be the permutation group generated by $\{g_v\} \cup \{h\}$. Note that the permutations in this set commute and have order 2. Let C denote the coset of G in G' which is not equal to G. As before, the permutations in C are in one-to-one correspondence with the cuts of H. A cut of size ℓ corresponds to a permutation with $2|E|r - (|E| - \ell)r = |E|r + \ell r$

cycles. Let $P(G)$ denote the output of #PÓLYAORBITS(G) and let N_ℓ denote the number of size-ℓ cuts of H. Then

$$P(G') - P(G) = \sum_{g \in C} k^{c(g)} = \sum_{\ell=0}^{j} N_\ell k^{|E|r+\ell r}, \text{ so}$$

$$\frac{P(G') - P(G)}{k^{|E|r+jr}} = N_j + \sum_{\ell=0}^{j-1} N_\ell k^{(\ell-j)r}.$$

Since $\sum_{\ell=0}^{j-1} N_\ell k^{(\ell-j)r}$ is non-negative and (from the definition of N_ℓ) is at most $2^{|V|} k^{-r}$ we have

$$N_j \leq \frac{P(G') - P(G)}{k^{|E|r+jr}} \leq N_j + 2^{|V|} k^{-r}.$$

Since $r = |V|^2$ and $k \geq 2$ and N_j is an integer,

$$N_j = \left\lfloor \frac{P(G') - P(G)}{k^{|E|r+jr}} \right\rfloor.$$

□

3.3 Counting subtrees of a tree

In this section, we will consider an exact orbit-counting problem that has a rather different flavour from the Pólya-theory problem of the previous section. We will see that this problem too is #P-complete. Thus, exactly counting orbits is computationally difficult, even in a seemingly simple setting.

Suppose that T is a tree with vertex set V and edge set E. We will let Ω_T be the set containing all (labelled) *subtrees* of T. That is, every element T' of Ω_T is a graph with vertex set V, some edge set $E' \subseteq E$, and at most one non-trivial connected component. (At most one connected component of T' will contain edges.) Let G_T be the permutation group acting on Ω_T which is induced by vertex permutations. As in Example 2.2, the orbits correspond to *unlabelled* subtrees of T. Jerrum and I [20] have shown that this orbit-counting problem, the problem #SUBTREES below, is #P-complete.

Name. #SUBTREES.

Instance. A tree T.

Output. The number of distinct (up to isomorphism) subtrees of T. That is, $|\Phi(\Omega_T, G_T)|$.

The proof that #SUBTREES is #P-complete is contained in [20]. The complexity of most variants of the problem is still unknown. For example, the status of the following problem is open.

Name. #SubForests.

Instance. A tree T.

Output. The number of distinct (up to isomorphism) subforests of T.

The constructions used in [20] involve trees with high-degree vertices, so it is also not clear whether the following problem is #P-complete for any constant $\Delta > 2$.

Name. #ΔTree-SubTrees.

Instance. A tree T in which every vertex has degree at most Δ.

Output. The number of distinct (up to isomorphism) subtrees of T.

We will conclude this section by briefly considering the following generalisation of #ΔTree-SubTrees.

Name. #ΔGraph-SubTrees.

Instance. A graph H in which every vertex has degree at most Δ.

Output. The number of distinct (up to isomorphism) subtrees of H.

Corollary 6 of [20] shows that #ΔGraph-SubTrees is in the complexity class $\mathrm{FP}^{\#\mathrm{P}}$. Informally, this is the class of functions which are "as easy" as #P. More formally, a function f is in $\mathrm{FP}^{\#\mathrm{P}}$ if it is polynomial-time Turing-reducible to a problem in #P. Thus, by the following lemma, which is proved in the appendix, #ΔGraph-SubTrees is complete in $\mathrm{FP}^{\#\mathrm{P}}$ for every fixed $\Delta \geq 5$.

Lemma 3.3 (Goldberg, Jerrum, Kelk) *For any fixed $\Delta \geq 5$, the problem #ΔGraph-SubTrees is #P-hard.*

3.4 Orbit-counting problems and the complexity class $\#\mathrm{P}_1$

In the previous section, we have seen that the problem of counting unlabelled subtrees of a given tree T is #P-complete. Now suppose that instead of having a particular n-vertex tree T as input, the input is just n and we are interested in counting *all* unlabelled trees with at most n vertices. We will consider the following computational problem.

Name. #Trees.

Instance. A positive integer n, expressed in unary.[3]

Output. The number of distinct (up to isomorphism) n-vertex trees.

[3] The reason that the input to #Trees is expressed in unary is that we are interested in knowing whether there is an algorithm for #Trees whose running time is bounded from above by a polynomial in n. Since there are exponentially many unlabelled n-vertex trees, an algorithm whose running time is bounded from above by a polynomial in $\log n$ would not even have enough time to write down the answer.

The problem #Trees can be viewed as an orbit-counting problem. In particular, it is the orbit-counting problem corresponding to input set $\{(\Omega_n, G_n)\}$ in which Ω_n is the set of n-vertex trees, and G_n is the group induced by vertex permutations (see Example 2.2).

#Trees can be solved in polynomial time using a generating function for the number of orbits. Once the generating function is given, its coefficients can be computed by dynamic programming. Harary and Palmer's book [23] contains a survey on using generating functions to do unlabelled enumeration. Their book gives a full treatment of the enumeration of unlabelled trees, following the work of Otter [43]. In order to illustrate the principles, we repeat a few of the details here. Let $T(x) = \sum_{n=1}^{\infty} T_n x^n$ be the generating function for *rooted* unlabelled trees. That is, T_n is the number of rooted unlabelled trees with n vertices. Pólya's theorem gives an expression[4] for $T(x)$ which can be manipulated to yield the recurrence

$$T_{n+1} = \frac{1}{n} \sum_{k=1}^{n} \left(\sum_{d|k} d\, T_d \right) T_{n-k+1}, \tag{3.1}$$

where the sum is over all divisors d of k. Using this formula, the coefficients $T_1, T_2, \ldots, T_n$ can be computed in polynomial time by dynamic programming. ("Dynamic programming" just means that the coefficients should be computed in the order $T_1, T_2, \ldots$. Note that a recursive algorithm would not complete in polynomial time unless a device such as a "memory function" is used.) Next, let $t(x) = \sum_{n=1}^{\infty} t_n x^n$ be the generating function for (unrooted) unlabelled trees. It can be shown that

$$t(x) = T(x) - \frac{1}{2}\left(T^2(x) - T(x^2)\right),$$

so the coefficients $t_1, t_2, \ldots, t_n$ can also be computed in polynomial time.

Now that we have seen a polynomial-time algorithm for #Trees, let us consider the following related problem from Example 2.2.

Name. #Graphs.

Instance. A positive integer n, expressed in unary.

Output. The number of distinct (up to isomorphism) n-vertex graphs.

There is no known generating function which would enable us to quickly solve #Graphs. In fact, there is no known polynomial-time algorithm (of any type) for this problem.

Both #Trees and #Graphs are examples of problems from the complexity class $\#\mathrm{P}_1$. The definition of this class is similar to the definition of #P. The only difference is that the input alphabet is now unary rather than binary. Valiant [50] has shown that $\#\mathrm{P}_1$ does contain complete problems.

[4] This expression was also discovered by Cayley.

Notably, Bertoni, Goldwurm and Sabadini have shown that counting strings of a given length in some context-free language is complete [4]. Nevertheless, no natural combinatorial problem is known to be complete for $\#\mathrm{P}_1$ and it seems unlikely that a problem such as #GRAPHS would be complete. Thus, at present, we seem to lack methods for quantifying the computational complexity of #GRAPHS and similar problems. This is an intriguing open question in the complexity theory of counting.

4 Approximate counting

Definition A *randomised approximation scheme* for a function $f : \{0,1\}^* \to \mathbb{N}$ is a probabilistic Turing machine that takes as input a pair $(x, \epsilon) \in \{0,1\}^* \times (0,1)$ and produces as output an integer random variable Y satisfying the condition $\Pr(e^{-\epsilon} \leq Y/f(x) \leq e^{\epsilon}) \geq 3/4$. Such an approximation scheme is said to be a *fully polynomial*[5] randomised approximation scheme (or FPRAS) if its running time is bounded from above by a polynomial in $|x|$ and ϵ^{-1}.

Thus, an algorithm for the approximate counting problem of Section 2.2 is an FPRAS if and only if its running time is bounded from above by a polynomial in the size of the description of the the input (Ω, G), and in ϵ^{-1}.

Clearly, there is an FPRAS for the problem #TREES, since this problem can be solved (exactly) in deterministic polynomial time (see Section 3.4). We will see in Section 5.6 that there is also an FPRAS for #GRAPHS. It is worth observing at this point that there are asymptotic enumerations of unlabelled graphs based on Pólya's theorem, but these do not seem to be strong enough to give an FPRAS. In particular, let U_n denote the number of unlabelled n-vertex graphs. Pólya showed that U_n is asymptotically equal to $2^{\binom{n}{2}}/n!$. Oberschelp [42] gave a more detailed formula for U_n with improved error terms. (See Chapter 9 of [23].) For example, he showed that there is a constant c such that

$$U_n \leq \frac{2^{\binom{n}{2}}}{n!} \left(1 + \frac{cn^2}{2^n}\right). \tag{4.1}$$

Equation 4.1 is sufficiently accurate when the desired error, ϵ, exceeds $cn^2/2^n$. However, it is not immediately clear how to approximate U_n when the error parameter ϵ is smaller. Note that it takes $\Omega(n!)$ time to apply Pólya's theorem directly and this can exceed $\mathrm{poly}(\epsilon^{-1})$ even when ϵ is too small for using Equation 4.1.

We will return to the problem #GRAPHS in Section 5.6, where we will describe an FPRAS. It is not known whether there are efficient approximate counting algorithms for the rest of the problems introduced in Section 3. Before we say more about these problems, we will look briefly at the complexity-theory context.

[5] The definitions that we use are taken from [12] but they are closely related to Karp and Luby's definitions from [33].

4.1 The complexity of approximate counting

From a complexity-theoretic point of view, *exactly* solving a #P-complete problem seems to be much more difficult than *approximately* solving it. The best way to illustrate this point is to introduce the notion of the "polynomial hierarchy". We will just state the relevant facts without giving details or definitions. Details can be found in [14] and [44]. The polynomial hierarchy contains an infinite sequence of complexity classes, $\Sigma_0^p, \Sigma_1^p, \ldots$. The class Σ_0^p is the same as the familiar class P and the class Σ_1^p is the same as NP. It is widely believed that all classes in the hierarchy are distinct. In particular, Σ_i^p is believed to be a proper subset of Σ_{i+1}^p. We can now state the relevance of the polynomial hierarchy — Toda [49] has shown that *every* problem in the entire polynomial hierarchy can be solved in polynomial time using an oracle for any #P-complete problem. Thus, informally, a #P-complete problem is "as hard as" the entire polynomial hierarchy. On the other hand, a result of Valiant and Vazirani [51] implies that every function in #P can be approximated (in the FPRAS sense) by a polynomial-time probabilistic Turing machine equipped with an NP oracle.[6] We can therefore conclude that the approximate counting problems in Sections 3.2 and 3.3 are "as easy as" NP, and we are interested in knowing whether they are easier.

Dyer, Greenhill, Jerrum, and I [12] recently studied the following notion of approximation-preserving reduction. Suppose $f, g : \{0,1\}^* \to \mathbb{N}$ are functions whose complexity (of approximation) we want to compare. An *approximation-preserving reduction* from f to g is a probabilistic oracle Turing machine M that takes as input a pair $(x, \epsilon) \in \{0,1\}^* \times (0,1)$, and satisfies the following three conditions: (i) every oracle call made by M is of the form (w, δ), where $w \in \{0,1\}^*$ is an instance of g and $0 < \delta < 1$ is an error bound satisfying $\delta^{-1} \leq \text{poly}(|x|, \epsilon^{-1})$; (ii) the Turing machine M meets the specification for being a randomised approximation scheme for f whenever the oracle meets the specification for being a randomised approximation scheme for g; and (iii) the run-time of M is polynomial in $|x|$ and ϵ^{-1}. If an approximation-preserving reduction from f to g exists we write $f \leq_{\mathrm{AP}} g$, and say that *f is AP-reducible to g*.

In [12], we identify a class of problems which are complete for #P with respect to AP-reducibility. It is unlikely that any of these problems has an FPRAS. In particular, if any such complete problem has an FPRAS then so does every problem in #P. This, in turn, would imply that RP = NP, which is unlikely.

We will not be using the complexity class RP after Section 4, but for completeness, we provide a definition. A decision problem (i.e., a problem with a "yes"/"no" answer) is in RP (see Chapter 11 of [44]) if there is a randomised

[6] This is Corollary 3.6 of [51]. Only a sketch of the proof appears in [51], but a detailed proof appears in Chapter 10 of [21]. For a related result, see [47].

polynomial-time algorithm which, for every "no" instance, answers "no" and for every "yes" instance, produces an output ("yes" or "no") which, on any given run, has probability at least $1/2$ of being "yes". The relationship between RP and the more familiar classes P and NP is given by $\mathrm{P} \subseteq \mathrm{RP} \subseteq \mathrm{NP}$. It is widely conjectured (for example, Chapter 7 of [21]) that $\mathrm{P} = \mathrm{RP}$, or at least that $\mathrm{RP} \neq \mathrm{NP}$.

4.2 Approximately automating Pólya theory

It is an intriguing open question whether #PólyaOrbits is complete for #P with respect to AP-reducibility. The most that we can say at this point is that a related problem (in which we work in a coset rather than in a group) is complete in this sense. The relationship between the new problem and #PólyaOrbits will be more clear if we first give a new definition of #PólyaOrbits, which is equivalent to the original definition by Pólya's theorem (Lemma 2.3). Recall that k is the size of the alphabet Σ in which the words are constructed.

Name. #PólyaOrbits.

Instance. $O(m)$ generators for a group G of permutations of $[m]$.

Output. $\sum_{g \in G} k^{c(g)}$.

We now describe the related problem, in which we sum permutations over a coset, rather than over the entire group.

Name. #CosetOrbits.

Instance. $O(m)$ generators for a group G' of permutations of $[m]$, $O(m)$ generators for a subgroup G of G' and a permutation $h \in G'$.

Output. $\sum_{g \in Gh} k^{c(g)}$.

Note that #PólyaOrbits corresponds to the special case of #CosetOrbits in which the coset Gh is a group. The following lemma implies that #CosetOrbits is unlikely to have an FPRAS, in which case coset decomposition cannot be used to give an FPRAS for #PólyaOrbits.

Lemma 4.1 #CosetOrbits *is complete for* #P *with respect to AP-reducibility.*

Proof Recall the problem #LargeCut from Section 3.2. Theorem 1 of [12] shows that #LargeCut is complete for #P with respect to AP-reducibility. Thus, it will suffice to show that #LargeCut $\leq_{\mathrm{AP}}$ #CosetOrbits. Let j and H be an instance of #LargeCut and let N_j denote the number of size-j cuts of H. Construct G', G and h as in the proof of Lemma 3.2. Now note that the output of #CosetOrbits corresponding to input (G', G, h), which we denote $\#\textsc{CosetOrbits}(G', G, h)$, is equal to the quantity $P(G') - P(G)$ in the notation of Lemma 3.2. Thus,

$$N_j = \left\lfloor \frac{\#\textsc{CosetOrbits}(G', G, h)}{k^{|E|r+jr}} \right\rfloor. \tag{4.2}$$

We conclude that a good approximation to $\#\textsc{CosetOrbits}(G', G, h)$ gives a good approximation to N_j. We will omit the details about how to choose the accuracy parameter δ in the reduction. If it were not for the floor function in (4.2), we could simply set $\delta = \epsilon$, since division by a constant preserves relative error. The discontinuous floor function could spoil the approximation when its argument is small. However, this is a technical problem and not a real difficulty. For a solution, see the proof of Theorem 3 of [12]. □

We have now shown that the approximation problem corresponding to #CosetOrbits is intractable, subject to the standard complexity-theoretic assumption that RP ≠ NP. It seems plausible that the approximation problem corresponding to #PólyaOrbits is also intractable, perhaps in the sense that it is also complete for #P with respect to AP-reducibility.[7] In Section 5.5 we will return to this problem and we will describe some special cases of #PólyaOrbits for which fully polynomial randomised approximation schemes are known. We close this section by mentioning a surprising fact. Although it is currently unknown whether #PólyaOrbits has an FPRAS for any fixed integer $k > 1$, Jerrum and I (Theorem 4 of [17] or Theorem 6 of [27]) have shown that if k is allowed to be any fixed rational *that is not an integer* then there is no FPRAS for #PólyaOrbits unless RP = NP. Our proof for the case in which k is not an integer sheds no light on the intriguing integer case.

5 Almost-uniform sampling

Definition An algorithm for the almost-uniform sampling problem in Section 2.2 is said to be a *fully polynomial* almost-uniform sampler if its running time is bounded from above by a polynomial in the size of the description of the input (Ω, G) and in $\log(\epsilon^{-1})$.

The notion of "fully polynomial almost-uniform sampling" is due to Jerrum, Valiant and Vazirani [31]. The particular definition that we use is based on the one in [11]. Since the running time of a fully polynomial almost-uniform sampler is bounded from above by a polynomial in the *logarithm* of ϵ^{-1} (rather than just by a polynomial in ϵ^{-1}), the output distribution D (see Section 2.2)

[7] Note that the reduction in Lemma 3.2 is not approximation preserving. In particular, approximations for $P(G')$ and $P(G)$ do not give an accurate approximation for $P(G')-P(G)$. For example, $e^{\epsilon}P(G') - e^{-\epsilon}P(G)$ can be much larger than $e^{\epsilon}(P(G') - P(G))$.

can be made very close to the uniform distribution $\mathcal{U}_{\Omega,G}$ at modest computational expense. For example, if ϵ is taken to be $e^{-|(\Omega,G)|}$, where $|(\Omega,G)|$ denotes the size of the input (Ω,G), then the variation distance between the two distributions is exponentially small in $|(\Omega,G)|$, even though the running time is only polynomial in $|(\Omega,G)|$. [8]

5.1 Almost-uniform sampling and approximate counting

Jerrum, Valiant and Vazirani [31] have shown that there is a close connection between almost-uniform sampling and approximate counting. In particular, for "self-reducible" combinatorial structures [46], a fully-polynomial almost-uniform sampler exists if and only if an FPRAS exists. We will not give a formal definition of "self-reducible" but intuitively it means that outputs corresponding to a given input can be expressed in terms of outputs corresponding to "smaller" inputs. That is, the family of combinatorial structures has an inductive definition. The techniques from [31] have been used to get similar results for some combinatorial structures that do not seem to be self-reducible (see [11]). Furthermore, Dyer and Greenhill [11] have extended the result to the (related, but larger) class of "self-partitionable" structures. Self-reducibility and self-partitioning do not seem to apply (in general) to orbit-counting problems and there is no known *general* connection between the (approximate) orbit-counting problem and the orbit sampling problem. We shall revisit this point briefly in Section 5.5. The reader is also referred to Jerrum's papers [25] and [27].

One situation in which "counting" technology can be used for sampling orbits is when generating functions for enumerating orbits can be efficiently evaluated. For example, Nijenhuis and Wilf [41] used Equation 3.1 (see Section 3.4) to obtain a polynomial-time algorithm for sampling rooted unlabelled trees. Their algorithm is given in Figure 1 (see also [48]). Note that this is an *exactly* uniform sampling algorithm — its output distribution is exactly the uniform distribution on orbits.

Nijenhuis and Wilf's approach was extended by Wilf [52], who gave a fully-polynomial almost-uniform sampler for the problem #TREES from Section 3.4. Once again, the output distribution of Wilf's algorithm is *exactly* uniform on orbits. Wilf's algorithm is also based on finding a recurrence for the coefficients of the relevant generating function. This approach to sampling has been sys-

[8]Note that a similarly demanding definition would not make sense in the context of an FPRAS. If we changed the definition of FPRAS (at the start of Section 4), demanding instead that the running time be bounded from above by a polynomial in $|x|$ and $\log(\epsilon^{-1})$, then finding an FPRAS for a problem would be as difficult as finding an exact algorithm. In particular, for many counting problems f, the quantity $f(x)$ is only exponentially large (as a function of $|x|$). Such problems could be solved exactly in polynomial time by running an FPRAS with $\epsilon \leq 1/(2f(x))$. The close connection between almost-uniform sampling and approximate counting (Section 5.1) indicates that these definitions (less demanding for FPRAS and more demanding for almost-uniform sampling) are the "right" ones.

1. Choose a pair (d, k) such that $k \in [1, n-1]$ and d divides k. The probability that the particular pair (d, k) is chosen should be $\frac{dT_{n-k}T_d}{(n-1)T_n}$.
2. Recursively choose T' uniformly at random (u.a.r.) from $\mathcal{R}_{n-k}$.
3. Recursively choose T'' u.a.r. from $\mathcal{R}_d$.
4. Make k/d copies of T'' and attach the root of each copy to the root of T'.
5. Let the root of T' be the root of the new n-vertex tree and output the new tree.

Figure 1: Let $\mathcal{R}_n$ be set of rooted unlabelled n-vertex trees. Suppose $n > 2$. The tree output by this algorithm of Nijenhuis and Wilf is equally likely to be any element of $\mathcal{R}_n$. The reason for this is given in Equation 3.1 — every n-vertex output comes up $n-1$ times in the following process. Choose k and d. Choose a d-vertex tree and an $n-k$-vertex tree. Connect these as described in the algorithm. Count the resulting n-vertex tree d times. The quantities T_i are computed using dynamic programming as in Section 3.4.

tematised by Flajolet, Zimmerman and Van Cutsem [13]. In their systematic approach, one specifies a set of structures using a formal grammar involving set, sequence and cycle constructions. Generating functions can be derived automatically from the specification, so uniform sampling can be done automatically using dynamic programming. The combinatorial structures studied in [13] are labelled structures, but the authors observe that similar principles can sometimes be used for sampling "unlabelled structures" (orbits). Jerrum and I have used their approach to sample some tree-like unlabelled structures in Section 4 of [19].

5.2 The orbit-sampling process

We will now describe a general Markov-chain approach for sampling orbits. The approach was proposed[9] by Jerrum [25]. It is essentially a random walk on the bipartite graph which corresponds to the orbit-counting lemma (Lemma 2.1). In particular, consider the bipartite graph in which the left-hand vertex set is a finite set Ω and the right-hand vertex set is a permutation group G acting on Ω. There is an edge between element $\alpha \in \Omega$ and permutation $g \in G$ if and only if $\alpha^g = \alpha$. The Markov chain $M(\Omega, G)$, which we

[9] Jerrum's description of the Markov chain was in terms of the Pólya-theory setting of Section 2.1. However, as Cameron has observed [7], the chain is applicable in the general orbit-counting setting.

refer to as the "orbit-sampling process", is essentially a random walk on this graph. In particular, the state space of $M(\Omega, G)$ is the set Ω. The transition probabilities from a state $\alpha \in \Omega$ are specified by the following two-step experiment:

1. Sample g uniformly at random (u.a.r.) from G_α.
2. Sample α' u.a.r. from $\mathrm{fix}(g)$.

The new state is α'. The chain $M(\Omega, G)$ is *ergodic* since every state α can be reached from every other in a single transition, by selecting the identity permutation in Step 1. (For Markov-chain definitions, see Chapter 6 of [22]). Let $\pi : \Omega \to [0,1]$ denote the stationary distribution of $M(\Omega, G)$. It is now straightforward to verify that $\pi(\alpha)$ is proportional to the degree of α in the bipartite graph. That is, $\pi(\alpha) = |G_\alpha|/|\Upsilon(\Omega, G)|$. We have thus established the following Lemma from [25]:

Lemma 5.1 *Let π be the stationary distribution of the Markov chain $M(\Omega, G)$. Then*

$$\pi(\alpha) = \frac{|G_\alpha|}{|\Upsilon(\Omega, G)|} = \frac{|G|}{|\alpha^G|\,|\Upsilon(\Omega, G)|} = \frac{1}{|\alpha^G|\,|\Phi(\Omega, G)|} \tag{5.1}$$

for all $\alpha \in \Omega$. in particular, π assigns equal probability to each orbit α^G.

The second equality in Equation 5.1 follows from Lagrange's Theorem which implies that $|G_\alpha| \times |\alpha^G| = |G|$ and the third follows from Lemma 2.1.

Since the stationary distribution of $M(\Omega, G)$ is uniform on orbits, a reasonable approach to the orbit-sampling problem is to simulate $M(\Omega, G)$ for a sufficient number of transitions (to get "close" to the stationary distribution) and then output the result. Two issues arise at this point:

1. Can the steps of $M(\Omega, G)$ be simulated efficiently, and
2. how many transitions have to be simulated before the chain is close to stationarity? In particular, how many transitions have to be simulated before almost-uniform sampling is achieved? (The definition of "almost-uniform sampling" is in Section 2.2.)

Both of these questions depend upon the specific input set $\mathcal{I}$ and the specific representation of the inputs in $\mathcal{I}$.

Let π be the stationary distribution of the Markov chain $M(\Omega, G)$. Let π_t be the distribution of $M(\Omega, G)$ after t transitions, when started in state α_0.

Definition The *mixing time* of $M(\Omega, G)$, given initial state α_0, is a function $\tau_{\alpha_0} : (0,1] \to \mathbb{N}$, from tolerances ϵ to simulation times, defined as follows: for each $\epsilon \in (0,1]$, let $\tau_{\alpha_0}(\epsilon)$ be the smallest t such that $\mathrm{d_{tv}}(\pi_{t'}, \pi) \le \epsilon$ for all $t' \ge t$. We define $\tau(\epsilon)$ to be the maximum of $\tau_{\alpha_0}(\epsilon)$ over all initial states $\alpha_0 \in \Omega$. $M(\Omega, G)$ is said to be *rapidly mixing* if and only if $\tau(\epsilon)$ is at most a polynomial in the size of the input (Ω, G) and in $\log(\epsilon^{-1})$.

Note that if $M(\Omega, G)$ is rapidly mixing, and each transition can be implemented in polynomial time, then $M(\Omega, G)$ is a fully-polynomial almost uniform sampler for orbits.

5.3 The orbit-sampling process and Pólya theory

Let $\mathcal{I}(\mathcal{G})$ be an input set in the Pólya-theory setting. Recall that each input $(\Sigma^m, \widehat{G})$ is represented as a set of $O(m)$ generators for G. Thus, the size of the input is bounded from above by a polynomial in m.

In this framework, Step 2 of each transition is computationally easy: to sample α' u.a.r. from $\mathrm{fix}(\hat{g})$, one just considers each of the $c(g)$ cycles of g and chooses one of the k alphabet symbols u.a.r. (see Section 2.1). However, Step 1 is apparently difficult. It is equivalent under randomised polynomial-time reductions to the *Setwise Stabiliser* problem, which includes *Graph Isomorphism* as a special case. There are, nevertheless, significant sets $\mathcal{G}$ of groups G for which a polynomial-time implementation exists. Luks has shown that p-groups—groups in which every element has order a power of p for some prime p—is an example of such a set [36]. For the remainder of this section, we will restrict our attention to input sets corresponding to sets $\mathcal{G}$ of permutation groups for which each transition can be implemented in polynomial time.

5.3.1 Negative Results Jerrum [25] asked whether the orbit-sampling process is rapidly mixing for the input set $\mathcal{I}(\mathcal{P})$. Subsequently [18], he and I showed that this is not the case. In particular, we constructed an infinite set $\mathcal{G}$ of permutation groups such that when the inputs $(\Sigma^m, \widehat{G})$ are chosen from $\mathcal{I}(\mathcal{G})$, the mixing time $\tau(1/3)$ of $M(\Sigma^m, \widehat{G})$ is exponential in m.

We will describe the construction (but not the proofs) here. Let k be the size of the fixed alphabet Σ. Let $\lambda = 1/k^2$. We will construct one group for each[10] pair $(l, n(l))$ where l and $n(l)$ are natural numbers satisfying

$$\left| \left(1 - \frac{(1+2\lambda)^l - (1-\lambda)^l}{(1+2\lambda)^l + 2(1-\lambda)^l} \right) - \frac{4 \ln 2}{n(l)} \right| \leq \frac{3}{n(l)^2}.$$

To construct the group $G_{l,n(l)}$, we let $H_{l,n(l)}$ denote the graph which is obtained from the complete graph on $n(l)$ vertices by subdividing each edge, inserting $l-1$ intermediate vertices of degree two. Thus, $H_{l,n(l)}$ is formed by applying the "l-stretch" operation of Jaeger, Vertigan and Welsh [24] to the complete graph $K_{n(l)}$. Let $V_{l,n(l)}$ and $E_{l,n(l)}$ denote the vertex and edge sets of $H_{l,n(l)}$ (respectively) and let $m_{l,n(l)}$ be $3\,|E_{l,n(l)}|$. We will construct a degree-$m_{l,n(l)}$ permutation group $G_{l,n(l)}$.

$G_{l,n(l)}$ acts on the set $K = \bigcup_{e \in E_{l,n(l)}} K_e$, which is the disjoint union of three-element sets K_e. Arbitrarily orient the edges of $H_{l,n(l)}$, so that each edge $e \in E_{l,n(l)}$ has a defined start-vertex e^- and end-vertex e^+. For $e \in E_{l,n(l)}$ and

[10] In [18] we prove that there are infinitely many such pairs.

$v \in V_{l,n(l)}$, let h_e be some fixed permutation that induces a 3-cycle on K_e and leaves everything else fixed and let g_v be the generator

$$g_v := \prod_{e:e^+=v} h_e \prod_{e:e^-=v} h_e^{-1}.$$

Finally, let $G_{l,n(l)}$ be $\langle g_v : v \in V_{l,n(l)} \rangle$, the group generated by $\{g_v\}$. Observe that the generators of the group commute and have order three, so each permutation $g \in G_{l,n(l)}$ can be expressed as a product

$$\prod_{v \in V_{l,n(l)}} {g_v}^{\sigma(v)},$$

where $\sigma : V \to \{0,1,2\}$. Thus, for every pair $(l, n(l))$, the group $G_{l,n(l)}$ is Abelian and every permutation $g \in G_{l,n(l)}$ (other than the identity) has order 3.

Let $\mathcal{G} = \{G_{l,n(l)}\}$. In [18], we showed that for any $\delta > 0$, the mixing time of the orbit-sampling process with input set $\mathcal{I}(\mathcal{G})$ satisfies $\tau(1/3) = \Omega(\exp(m(G)^{1/(4+\delta)}))$. Thus, the orbit-sampling process mixes slowly for an infinite set of Abelian 3-groups.

We will not describe the slow-mixing proof here, but the high-level picture is as follows: We can identify two types of permutation $g \in G$ such that, when the chain is in the stationary distribution, the permutation g selected in Step 1 is quite likely to have type 1 and also quite likely to have type 2. On the other hand, it takes the chain a long time to move from a permutation of one type to a permutation of the other type, and this implies slow mixing.

5.3.2 Positive results Despite the slow-mixing result of the previous section, Jerrum [25] has identified two sets of permutation groups for which the orbit-sampling chain is rapidly mixing.

1. $\mathcal{G}$ is the set of symmetric groups, as in Example 2.4.
2. $\mathcal{G}$ is the set of all cyclic groups (all groups which are generated by a single permutation).

Jerrum showed that the orbit-sampling process is rapidly mixing in both cases, so this process provides a fully-polynomial almost-uniform sampler in these cases.

To illustrate the ideas, we will consider the second case. Let G be a degree-m cyclic group and consider the Markov chain $M(\Sigma^m, \widehat{G})$.

As before, let π be the stationary distribution of $M(\Sigma^m, \widehat{G})$, and let π_t be the distribution after t transitions, starting from state α_0. A (Markovian) *coupling* for $M(\Sigma^m, \widehat{G})$ is a stochastic process (α_t, β_t) on $\Sigma^m \times \Sigma^m$ such that each of (α_t) and (β_t), considered marginally, is a faithful copy of $M(\Sigma^m, \widehat{G})$. In order to prove that $M(\Sigma^m, \widehat{G})$ is rapidly mixing, we want to construct a coupling in which the moves of (α_t) and (β_t) are correlated, so that (α_t) and (β_t) *coalesce* rapidly, ensuring that $\alpha_t = \beta_t$ for all sufficiently large t. The coupling lemma (see, for example, Aldous [1]) says that if β_0 is chosen from π then

$$\mathrm{d_{tv}}(\pi_t, \pi) \leq \Pr[\alpha_t \neq \beta_t].$$

Let $\mathbf{1}$ denote the identity permutation. Let g_i denote the permutation chosen in Step 1 of the i'th transition of $M(\Sigma^m, \widehat{G})$. Jerrum showed that there is a constant ϵ and a polynomial $p(m)$ such that for every permutation $g \in G$ $\Pr(g_{p(m)} = \mathbf{1} \mid g_1 = g) \geq \epsilon$.

Given this fact, the mixing time can be bounded via a straightforward coupling: Let the two copies run independently until they reach a transition during which they both select the identity during Step 1. After that, run the copies together, keeping the second copy in the same state as the first. The probability that coupling has not occurred by time τ is $\exp(-\Omega(\tau/p(m)))$, so the chain is rapidly mixing.

5.4 Open questions regarding the orbit-sampling process

As we observed in the previous section, when the set Ω consists of words in the Pólya-theory framework and the group G is cyclic, the orbit-sampling process visits the identity permutation often, and this implies that it mixes rapidly. Similarly, when the group is the symmetric group, the process visits the word $\alpha = 00\cdots0$ often, and it mixes rapidly. I am not aware of any other rapid-mixing results for the orbit-sampling process. It would be interesting to identify a non-trivial input set for which the chain is rapidly mixing, but for some other reason. As a test case, we might ask whether it is rapidly mixing when orbits correspond to unlabelled 2-regular graphs. However, note that unlabelled 2-regular graphs can easily be sampled directly using the connection to integer partitions. See [35].

Cameron illustrated the orbit-sampling process in his textbook [7] by describing the case in which orbits represent unlabelled graphs (Example 2.2). In this case, Step 1 of the process corresponds to *Graph Isomorphism,* which we do not know how to solve in polynomial time. Nevertheless, as Cameron observes, there are good heuristics for graph isomorphism (for example, McKay's `nauty` [37]), so implementing the transitions may not represent a serious practical difficulty. It is worth recording the fact that we do not know whether the orbit-sampling process is rapidly mixing for unlabelled graphs. Probably it is. Since the identity permutation is visited often, a proof along the lines of the one sketched in Section 5.3.2 may work. However, as far as I know, nobody has proved this. In particular, even though it is clear that the identity permutation is visited often in the stationary distribution, it is not known whether there are some "bad" starting points from which it takes a long time to reach the identity. It would also be good to know whether the process is rapidly-mixing when orbits correspond to (unlabelled) bounded-degree graphs. In this case, the transitions of the process can be efficiently implemented.

5.5 Approximate counting revisited

There is no known general connection between the problem of approximately counting orbits and the orbit sampling problem (see Section 5.1). This is true even if we restrict attention to the Pólya-theory framework of Section 2.1. Nevertheless, in the Pólya-theory setting, the orbit-sampling *process* can be used for approximate counting.

Recall that

$$\Upsilon(\Omega, G) = \{(\alpha, g) \mid \ \alpha \in \Omega \text{ and } g \in G \text{ and } \alpha \in \mathrm{fix}(g) \ \}.$$

Definition A *fully-polynomial almost-uniform* Υ*-sampler* for an input set $\mathcal{I}$ is an algorithm which takes an input $(\Omega, G) \in \mathcal{I}$ and an accuracy parameter $\epsilon \in (0, 1]$ and outputs a random variable. Typically, the output is a member of $\Upsilon(\Omega, G)$. In particular, the variation distance between the output distribution of the algorithm and the uniform distribution on $\Upsilon(\Omega, G)$ should be at most ϵ. Furthermore, the running time of the algorithm should be bounded from above by a polynomial in the size of the description of the input and in $\log(\epsilon^{-1})$.

If we run the orbit-sampling process, and observe the pair (α', g) at the end of each transition, then the stationary distribution of the process is uniform on $\Upsilon(\Omega, G)$ (see Lemma 5.1). Thus, the process is a fully-polynomial almost-uniform Υ-sampler for an input set $\mathcal{I}$ if and only if it is rapidly mixing for $\mathcal{I}$.

Now let $\mathcal{I}(\mathcal{G})$ be an input set in the Pólya-theory setting. The following lemma is due to Jerrum.

Lemma 5.2 *[25] If there is a fully-polynomial almost-uniform* Υ*-sampler for* $\mathcal{I}(\mathcal{G})$ *then there is an FPRAS for the corresponding orbit-counting problem.*

Together with Jerrum's rapid-mixing results from Section 5.3.2, Lemma 5.2 implies that the problem #PÓLYAORBITS has an FPRAS if the group G is required to be cyclic or to be a symmetric group. We will not include the proof of the lemma but it will be useful to outline the key ideas, which are frequently used in the "Markov Chain Monte Carlo" area. Note that the proof in [25] uses slightly different definitions, but a general treatment, with definitions similar to ours can be found in [11]. First, since $|\widehat{G}|$ can be computed exactly in polynomial time, Lemma 2.1 implies that it suffices to approximate $|\Upsilon(\Sigma^m, \widehat{G})|$. In this approximation, the self-reducibility in the group structure can be exploited. In particular, it suffices to estimate m ratios of the form

$$\frac{|\Upsilon(\Sigma^m, \widehat{G}_{i-1})|}{|\Upsilon(\Sigma^m, \widehat{G}_i)|} \tag{5.2}$$

where

$$G_j = \{g \in G \mid \ell^g = \ell \text{ for all } \ell < j\}$$

and $i \in \{1, \ldots, m\}$. $|\Upsilon(\Sigma^m, \widehat{G}_m)|$ can be calculated exactly (it is k^m) and this can be multiplied by all of the ratios to yield $|\Upsilon(\Sigma^m, \widehat{G}_0)|$, which is the desired quantity. The ratio in Equation 5.2 can be estimated by sampling from $\Upsilon(\Sigma^m, \widehat{G}_{i-1})$ and checking how many of the samples are in $\Upsilon(\Sigma^m, \widehat{G}_i)$.

Lemma 5.2 tells us that approximate counting is as easy as almost-uniformly sampling from $\Upsilon(\Sigma^m, \widehat{G})$ but it is not known whether the converse is true. In particular, $\Upsilon(\Sigma^m, \widehat{G})$ does not seem to be "self-partitionable" in the sense of [11]. It is easy to see that the set $\Upsilon(\Sigma^m, \widehat{G})$ can be described inductively by breaking $\widehat{G}$ into cosets. However, the problem is that the natural "parts" are cosets rather than groups, and we already know from Lemma 4.1 that approximately counting is difficult over cosets.

In particular, a natural method for sampling from $\Upsilon(\Sigma^m, \widehat{G}_i)$ would be to use counting estimates to determine the relative weight of each coset of $\Upsilon(\Sigma^m, \widehat{G}_{i+1})$, then select a coset (with the appropriate probability) and recursively sample from the coset. But this approach is unlikely to lead to an efficient algorithm because of Lemma 4.1.

5.6 Other orbit-sampling methods

5.6.1 Wormald's Method As in Example 2.2, let Ω_n be the set of all n-vertex graphs and let G_n be the permutation group acting on Ω_n which is induced by vertex permutations. The orbits of Ω_n under G_n correspond to unlabelled n-vertex graphs. Let $\mathcal{I}$ be the input set $\{(\Omega_n, G_n)\}$. The input (Ω_n, G_n) will be represented by the positive integer n, encoded in unary, as in the problem #GRAPHS. It is unknown whether the orbit-sampling process is rapidly mixing for $\mathcal{I}$. Nevertheless, there is a fully-polynomial almost-uniform Υ-sampler for $\mathcal{I}$. Thus, by Lemma 5.2, there is also an FPRAS for $\mathcal{I}$.[11] The Υ-sampler is due to Wormald [55] and uses the "rejection sampling" method, which is a frequently-used and powerful tool for sampling.

In order to simplify the description of Wormald's algorithm, we introduce the following notation: For every permutation g of a set Ω, let

$$\Upsilon(\Omega, g) = \{(\alpha, g) \mid \ \alpha \in \Omega \text{ and } \alpha \in \text{fix}(g)\ \}.$$

Thus, $\Upsilon(\Omega, G) = \bigcup_{g \in G} \Upsilon(\Omega, g)$.

First, suppose that we could estimate $|\Upsilon(\Omega_n, g)|$ and $|\Upsilon(\Omega_n, G_n)|$. Then we could sample from $\Upsilon(\Omega_n, G_n)$ using the following algorithm of Dixon and Wilf [9]:[12]

[11] In order to apply Lemma 5.2, we are implicitly using the fact that $\mathcal{I}$ can be encoded in the Pólya-theory setting. See Example 2.5.

[12] Dixon and Wilf's algorithm is more sophisticated than the one that we describe here. In particular, they show that the probabilities in Step 2 are identical for permutations

1. Input n
2. Choose $g \in G_n$ with probability $\frac{|\Upsilon(\Omega_n, g)|}{|\Upsilon(\Omega_n, G_n)|}$.
3. Choose (α, g) u.a.r. from $\Upsilon(\Omega_n, g)$.

Step 3 of the algorithm is easily implemented — it corresponds to Step 2 of the orbit-sampling process. The main problem is that we do not know how to estimate $|\Upsilon(\Omega_n, G_n)|$. Wormald [55] uses *rejection sampling* to avoid doing this estimation. The basic idea of rejection sampling is as follows. It may be too difficult to sample from a given desired distribution. So what the user does instead is to sample from some other (more tractable) distribution. Imagine the desired distribution as being "scaled down" so that it fits underneath the more tractable distribution. To draw a sample from the desired distribution, the user first draws a sample from the more tractable distribution. The user then uses the sample to determine the probability with which the more tractable distribution over-represents this sample (relative to the "scaled down" desired distribution). With this probability, the sample is rejected (and the value $\perp$ is output instead). Otherwise, the sample is output. The method is useful when it is easy to determine the probability with which a given sample should be rejected (so rejection is fast) and, furthermore, the overall rejection probability is low (so the variation distance between the output distribution of the algorithm and the desired distribution is small).

We will now describe Wormald's algorithm. To simplify the description, we will first omit the accuracy parameter, ϵ, from the input. After we have described the algorithm, we will bound the variation distance between the output distribution of the algorithm and the uniform distribution on $\Upsilon(\Omega_n, G_n)$. We will then say how to modify the algorithm to reduce the variation distance to any desired quantity ϵ. The outline of the algorithm is as follows, where $\mathbf{1}$ denotes the identity permutation (This is a slight abuse of notation, since we use the single symbol $\mathbf{1}$, but when the input is n, we mean the identity permutation on Ω_n.) We will use the symbol $p_{g,n}$ to represent the probability with which permutation g is chosen (so $\sum_g p_{g,n} = 1$). Appropriate choices for $p_{g,n}$ will be discussed below.

1. Input n
2. Choose g with probability $p_{g,n}$.
3. Choose (α, g) u.a.r. from $\Upsilon(\Omega_n, g)$.
4. With probability $\frac{p_{\mathbf{1},n}}{|\Upsilon(\Omega_n, \mathbf{1})|} \frac{|\Upsilon(\Omega_n, g)|}{p_{g,n}}$ output (α, g); otherwise output $\perp$.

in the same conjugacy class, and by breaking G_n into conjugacy classes, they show how to implement Step 2 in polynomial time *on average* provided the value of $|\Upsilon(\Omega_n, G_n)|$ is known. Details can be found in [9].

Clearly, we will need to choose the probabilities $p_{g,n}$ in such a way that Criterion 1 (below) is satisfied (so that Step 4 can be implemented). We will also choose the probabilities in such a way that Criteria 2 and 3 are satisfied, so that the algorithm runs in polynomial time.

Criterion 1: The probabilities $p_{g,n}$ must be chosen so that

$$\frac{p_{\mathbf{1},n}}{|\Upsilon(\Omega_n, \mathbf{1})|} \frac{|\Upsilon(\Omega_n, g)|}{p_{g,n}} \leq 1.$$

Criterion 2: The probabilities $p_{g,n}$ must be chosen so that Step 2 can be implemented in polynomial time[13].

Criterion 3: The probabilities $p_{g,n}$ must be chosen so that Step 4 can be implemented in polynomial time.

It is easy to check that the probability that any given pair (α, g) from $\Upsilon(\Omega_n, G_n)$ is output is $p_{\mathbf{1},n} \,/\, |\Upsilon(\Omega_n, \mathbf{1})|$. With the remaining probability, which we denote π, the algorithm outputs $\perp$. It is now straightforward to verify that the total variation distance between the output distribution of the algorithm and the uniform distribution on $\Upsilon(\Omega_n, G_n)$ is π. Note that the rejection probability is 0 whenever $g = \mathbf{1}$. Thus, $\pi \leq 1 - p_{\mathbf{1},n}$.

If we wish to have an upper bound ϵ on the total variation distance, then we simply run the algorithm for $\lceil \log(\epsilon)/\log(1-p_{\mathbf{1},n}) \rceil$ iterations. If the output is *always* $\perp$ (for every iteration) then we output $\perp$. Otherwise, we output the first member of $\Upsilon(\Omega_n, G_n)$ which is output by an iteration. We get a fully-polynomial almost-uniform Υ-sampler as long as the total number of times that we run the algorithm is bounded from above by a polynomial in n and $\log(\epsilon^{-1})$. Since

$$\log(\epsilon)/\log(1 - p_{\mathbf{1},n}) = \log(\epsilon^{-1})/\log((1 - p_{\mathbf{1},n})^{-1}),$$

this follows from Criterion 4.

Criterion 4: The probabilities $p_{g,n}$ must be chosen so that, for some positive constant c and every n, we have $p_{\mathbf{1},n} \geq n^{-c}$.

Wormald [55] showed how to choose the probabilities $p_{g,n}$ so that these criteria are met. Thus, he gave a fully-polynomial almost-uniform Υ-sampler for unlabelled graphs.

[13] Note that choosing $p_{g,n} = \frac{|\Upsilon(\Omega_n, g)|}{|\Upsilon(\Omega_n, G_n)|}$ would make Wormald's algorithm equivalent to Dixon and Wilf's. Thus, it would satisfy all criteria except Criterion 2.

5.6.2 Extending Wormald's Method Wormald's method can easily be expressed in the general orbit-sampling framework. As before, $p_{g,G}$ denotes the probability with which permutation g is chosen, so $\sum_g p_{g,G} = 1$.

1. Input (Ω, G)
2. Choose $g \in G$ with probability $p_{g,G}$.
3. Choose (α, g) u.a.r. from $\Upsilon(\Omega, g)$.
4. With probability $\frac{p_{\mathbf{1},G}}{|\Upsilon(\Omega,\mathbf{1})|}\frac{|\Upsilon(\Omega,g)|}{p_{g,G}}$ output (α, g); otherwise output $\perp$.

The analogue of Criterion 4 states that there is a positive constant c such that for every possible input (Ω, G), we must have $p_{\mathbf{1},G} \geq m(G)^{-c}$. On the other hand, the analogue of Criterion 1 implies

$$\frac{|\Upsilon(\Omega, \mathbf{1})|}{p_{\mathbf{1},G}} \geq \frac{|\Upsilon(\Omega, g)|}{p_{g,G}}, \tag{5.3}$$

which implies

$$p_{\mathbf{1},G} \leq \frac{|\Upsilon(\Omega, \mathbf{1})|}{|\Upsilon(\Omega, G)|}.$$

Thus, we cannot simultaneously satisfy the two criteria unless there is a positive constant c such that for every possible input (Ω, G),

$$m(G)^{-c} \leq \frac{|\Upsilon(\Omega, \mathbf{1})|}{|\Upsilon(\Omega, G)|}. \tag{5.4}$$

In other words, we cannot use Wormald's method unless the the part of $\Upsilon(\Omega, G)$ which corresponds to the identity permutation accounts for at least a polynomial fraction of $\Upsilon(\Omega, G)$.

Several natural sets of permutation groups satisfy Equation 5.4. For example, Wormald has shown [55] how to use the method to efficiently sample unlabelled r-regular graphs for $r \geq 3$.

5.6.3 Other possibilities Not much is known about how to sample orbits when the input set does not satisfy Equation 5.4. We have just seen that Wormald's method relies on the identity permutation having a large weight (in the sense that Equation 5.4 must be satisfied). One of the two positive results in Section 5.3.2 (the result showing that the orbit-sampling process is rapidly mixing for cyclic groups) also relies on this fact.

Jerrum and I [19] considered the following orbit-sampling problem, which we chose specifically because Equation 5.4 does not hold.

Example 5.3 *Let Δ be any fixed constant. For any multigraph H with degree at most Δ, the* degree sequence *of H is a sequence $\mathbf{n} = n_0, \ldots, n_\Delta$, where n_i denotes the number of vertices of H with degree i. Let $\Omega_\mathbf{n}$ be the set of all n-vertex connected multigraphs with degree sequence $\mathbf{n}$. Let $G_\mathbf{n}$ be the permutation group acting on $\Omega_\mathbf{n}$ which is induced by vertex permutations. As in Example 2.2, the orbits correspond to isomorphism classes. That is, the orbits of $\Omega_\mathbf{n}$ under $G_\mathbf{n}$ correspond to the unlabelled connected multigraphs with degree sequence $\mathbf{n}$.*

In [19], we gave a fully polynomial almost-uniform sampler for this orbit-sampling problem (sampling unlabelled connected multigraphs with a given (bounded) degree sequence). Unfortunately, our solution does not contain any new methods — it is really a combination of the methods that have already been described here. Discovering new methods for sampling orbits, particularly methods which do not require Equation 5.4 remains an interesting challenge.

Our algorithm for sampling unlabelled connected multigraphs is based on the following idea. Every unlabelled connected multigraph H is associated with a unique "core"[14] which has no vertices of degree 1 or 2. To randomly generate a multigraph H, the algorithm first generates the core of H and then extends the core by adding trees and chains of trees to obtain H.

The algorithm for generating the core is described using the configuration model of Bender and Canfield [3], Bollobás [6] and Wormald [53]. A configuration (for a given degree sequence) is a labelled combinatorial structure which can be viewed as a refinement of a multigraph with the degree sequence. For any given degree sequence, the orbits of all configurations (with respect to the appropriate permutation group) correspond to the unlabelled multigraphs with the degree sequence. Since the degree sequence of the core has no vertices of degree 1 or 2, a typical core does not have many symmetries and Equation 5.4 is satisfied. (This follows from an extension of Bollobás's analysis of unlabelled regular graphs [5].) Thus, the algorithm uses Wormald's method to generate the core. (If the core is not connected, it is rejected. The fact that this does not happen too often follows from another result of Wormald [54].)

After generating the core of the random multigraph, the algorithm extends the core by adding trees and chains of trees. This part of the algorithm is based on the generating-function approach illustrated in Section 5.1.

It is an open problem to sample unlabelled multigraphs given a *general* degree sequence (in which degrees need not be bounded from above by a constant). Our method is not applicable when the degrees are unbounded. In fact, the problem with unbounded degrees seems to be difficult even in the labelled case (see [29, 39, 10]).

[14] For other uses of the "core" idea, see Zhan [56].

6 A related problem: Listing orbits

Consider the following computational problem, which fits into the framework of Section 2.2.

Definition *The orbit-listing problem:* Given an input $(\Omega, G) \in \mathcal{I}$, output exactly one member of each orbit in $\Phi(\Omega, G)$.

There is a vast literature on the problem of listing orbits. The reader is referred particularly to McKay's paper [38] which introduces a new method and also explains the connection between various methods which are used in practice. Further work along these lines can be found in [34]. In this survey we will restrict our attention to *polynomial delay* listing, which is not mentioned in these works.

The notion of "polynomial delay" is due to Johnson, Yannakakis and Papadimitriou [32]. A listing algorithm has polynomial delay if and only if the delay (in time-steps) between each pair of consecutive outputs is bounded from above by a polynomial (in the input size).

When the permutation group is trivial (so the orbits are in one-to-one correspondence with the elements of Ω), listing can be shown to be strictly less difficult than sampling [16], in the sense that the existence of a fully-polynomial almost-uniform sampling algorithm for a given input set implies the existence of a (randomised) polynomial-delay listing algorithm (but not vice-versa). It is not known whether such a result holds for arbitrary input sets, but the idea has been used for at least one non-trivial orbit sampling problem. In particular, Dixon and Wilf [9] suggested using a sampling algorithm for unlabelled graphs in order to list them. Using this idea, one can combine Wormald's unlabelled-graph sampling algorithm from Section 5.6.1 with Babai and Kučera's canonical labelling algorithm [2] to obtain a (randomised) polynomial-delay algorithm for listing unlabelled graphs [15, 16]. The duplicate-elimination contained in this algorithm requires each of the (exponentially-many) orbits to be stored. However, it turns out that there is also a deterministic polynomial-space polynomial-delay algorithm for listing unlabelled graphs [15, 16].

It is not known whether there is a polynomial-delay listing algorithm for listing orbits in the general Pólya-theory framework (i.e., for input set $\mathcal{I}(\mathcal{P})$). It would be interesting to know more about this question, and about its connection to the corresponding orbit-sampling problem.

Acknowledgements

This work was supported by the EPSRC Research Grant GR/M96940 and by the IST Programme of the EU under contract numbers IST-1999-14186 (ALCOM-FT) and IST-1999-14036 (RAND-APX). I am grateful to Mark Jerrum for many useful discussions and collaborations in this area and also for helpful comments on an earlier draft of this article. I am also grateful to the referee for helpful comments.

7 Appendix: Proof of Lemma 3.3

The proof is by reduction from the problem #CubicHam, which was shown to be #P-complete by Jerrum [26].

Name. #CubicHam.

Instance. A graph H in which every vertex has degree at most 3.

Output. The number of Hamiltonian paths in H.

Proof Let F_n be the graph with vertex set $\{c_{i,j} \mid i \in [n], j \in [2n+2]\}$ and edge set

$$\{(c_{i,0}, c_{i',0}) \mid i' = i+1 \pmod{n}\} \cup \bigcup_i \{(c_{i,j}, c_{i,j'}) \mid j' = j+1 \pmod{2n+2}\}.$$

Thus, F_n consists of a "central" length-n cycle $c_{0,0}, \ldots, c_{n-1,0}$ and, off of each vertex $c_{i,0}$ in the cycle, there is a length-$(2n+2)$ cycle $c_{i,0}, \ldots, c_{i,2n+1}$, which we refer to as a "petal". For every $j \in [n+1]$, let $F'_n[j]$ be the graph obtained from F_n by deleting edges $(c_{0,j}, c_{0,j+1})$ and $(c_{0,j+1}, c_{0,j+2})$. Thus, $F'_n[j]$ is obtained from F_n by removing two adjacent edges from a petal. The shortest path from the central cycle to the two deleted edges is the length-j path from $c_{0,0}$ to $c_{0,j}$. Let F'_n be the union of $n+1$ disjoint graphs, the jth of which is isomorphic to $F'_n[j]$.

Let H be an instance of #CubicHam with vertex set $V = \{v_{0,0}, \ldots, v_{n-1,0}\}$. Let H' be the graph with vertex set $V \cup \{v_{i,j} \mid i \in [n], j \in \{1, \ldots, 2n+1\}\}$ which is constructed from H by adding the edges in

$$\bigcup_i \{(v_{i,j}, v_{i,j'}) \mid \; j' = j+1 \pmod{2n+2} \text{ and } j \neq i+1\}.$$

Roughly, H' is formed from H by attaching petals, but the $i+1$st edge is deleted from the ith petal.

For any graph Γ, let $N(\Gamma)$ denote the number of distinct (up to isomorphism) subtrees of Γ. We claim that

$$N(H' \cup F_n) - N(H' \cup F'_n) = N(F_n) - N(F'_n) - \#\text{CubicHam}(H),$$

which completes the proof.

To see why the claim is true, note that to form a subtree of F_n, one must delete an edge $(c_{i,0}, c_{i',0})$. Also, for each $i \in [n]$, one must delete an edge $(c_{i,j}, c_{i,j'})$. If one stops at this point, then the subtree is not represented in $N(F'_n)$, but if any further edges are deleted, then the subtree is represented in $N(F'_n)$. Now we want to know how many subtrees in $N(F_n) - N(F'_n)$ are subtrees of H' and this turns out to be the number of Hamiltonian paths in H. $\square$

References

[1] D. Aldous, Random walks on finite groups and rapidly mixing Markov chains, *Séminaire de Probabilités XVII 1981/1982*, (ed. A. Dold and B. Eckmann), *Springer-Verlag Lecture Notes in Mathematics*, **986** Springer-Verlag, (1983). 243–297.

[2] L. Babai and L. Kučera, Canonical labeling of graphs in linear average time, in *Proceedings 20th IEEE Symposium on Foundations of Computer Science* (1979), 39-46.

[3] E. A. Bender and E. R. Canfield, The asymptotic number of labelled graphs with given degree sequences, *J. Combin. Theory Ser. A* **24** (1978), 296–307.

[4] A. Bertoni, M. Goldwurm and N. Sabadini, The complexity of computing the number of strings of given length in context-free languages, Rapporto Interno n. 26/88, Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, via Moretta da Brescia, 9-I 20133 Milano, Italy, 1988.

[5] B. Bollobás, The asymptotic number of unlabelled regular graphs, *J. London Math. Soc.* **26** (1982), 201–206.

[6] B. Bollobás, Almost all regular graphs are Hamiltonian, *European J. Combin.* **4** (1983), 97–106.

[7] P. Cameron, *Permutation Groups*, *London Mathematical Society Student Texts*, **45** Cambridge University Press, (1999).

[8] N. G. De Bruijn, Pólya's theory of counting, *Applied Combinatorial Mathematics*, (ed. E.F. Beckenbach), John Wiley and Sons, (1964).

[9] J. D. Dixon and H. S. Wilf, The random selection of unlabeled graphs, *J. Algorithms* **4** (1983), 205–213.

[10] M. Dyer and C. Greenhill, Polynomial-time counting and sampling of two-rowed contingency tables, *Theoret. Comput. Sci.* **246** (2000), 265–278.

[11] M. Dyer and C. Greenhill, Random walks on combinatorial objects, *Surveys in Combinatorics*, (ed. J.D. Lamb and D.A. Preece), *London Mathematical Society Lecture Note Series*, **267** Cambridge University Press, (1999). 101–136.

[12] M. Dyer, C. Greenhill, L.A. Goldberg and M. Jerrum, On the relative complexity of approximate counting problems, in *Proceedings of APPROX* Springer Lecture Notes in Computer Science, **1913** (2000), 108–119.

[13] P. Flajolet, P. Zimmerman and B. Van Cutsem, A calculus for the random generation of labelled combinatorial structures, *Theoret. Comput. Sci.* **132** (1994), 1–35.

[14] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, (1979).

[15] L. A. Goldberg, Efficient algorithms for listing unlabeled graphs, *J. Algorithms* **13** (1992), 128–143.

[16] L.A. Goldberg, *Efficient Algorithms for Listing Combinatorial Structures*, Cambridge University Press, (1993).

[17] L.A. Goldberg, Automating Pólya theory: the computational complexity of the cycle index polynomial, *Inform. and Comput.* **105(2)** (1993), 268–288.

[18] L. A. Goldberg and M. Jerrum, The "Burnside process" converges slowly, in *Randomization and Approximation Techniques in Computer Science, Proceedings of RANDOM 1998* (ed. M. Luby, J. Rolim and M. Serna), Springer Lecture Notes in Computer Science, **1518** (1998), 331–345.

[19] L.A. Goldberg and M. Jerrum, Randomly sampling molecules, *SIAM J. Comput.* **29(3)** (1999), 834–853.

[20] L.A. Goldberg and M. Jerrum, Counting unlabelled subtrees of a tree is #P-complete, *LMS J. Comput. Math.* **3** (2000), 117-124.

[21] O. Goldreich, *Introduction to Complexity Theory*, *Lecture Notes Series of the Electronic Colloquium on Computational Complexity*, (1999). `http://www.eccc.uni-trier.de/eccc-local/ECCC-LectureNotes/`

[22] G.R. Grimmett and D.R. Stirzaker, *Probability and Random Processes, Second Edition*, Oxford University Press, (1992).

[23] F. Harary and E.M. Palmer, *Graphical Enumeration*, Academic Press, (1973).

[24] F. Jaeger, D.L. Vertigan and D.J.A. Welsh, On the computational complexity of the Jones and Tutte polynomials, *Math. Proc. Cambridge Philos. Soc.* **108** (1990), 35–53.

[25] M. Jerrum, Uniform sampling modulo a group of symmetries using Markov chain simulation, *Expanding Graphs*, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, **10** (ed. J. Friedman), AMS, 37–47. (1993).

[26] M. Jerrum, Counting trees in a graph is #P-complete, *Inform. Process. Lett.* **51(3)** (1994), 111–116.

[27] M. Jerrum, Computational Pólya theory, *Surveys in Combinatorics*, *London Mathematical Society Lecture Note Series*, **218** Cambridge University Press, (1995). 103–118.

[28] M. Jerrum, Mathematical foundations of MCMC, *Probabilistic Methods for Algorithmic Discrete Mathematics*, (ed. M. Habib, C. McDiarmid, J. Ramirez-Alfonsin and B. Reed), Springer, (1998). 116–165.

[29] M. Jerrum and A. Sinclair, Fast uniform generation of regular graphs, *Theoret. Comput. Sci.* **73** (1990), 91–100.

[30] M. Jerrum and A. Sinclair, Polynomial-time approximation algorithms for the Ising model, *SIAM J. Comput.* **22** (1993), 1087–1116.

[31] M.R. Jerrum, L.G. Valiant and V.V. Vazirani, Random generation of combinatorial structures from a uniform distribution, *Theoret. Comput. Sci.* **43** (1986), 169–188.

[32] D.S. Johnson, M. Yannakakis and C.H. Papadimitriou, On generating all maximal independent sets, *Inform. Process. Lett.* **27** (1988), 119–123.

[33] R.M. Karp and M. Luby, Monte-Carlo algorithms for enumeration and reliability problems, in *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science* (1983), 56–64.

[34] A. Kerber, *Applied Finite Group Actions, 2nd Edition*, Springer-Verlag, (1999).

[35] D.L. Kreher and D.R. Stinson, *Combinatorial Algorithms: Generation, Enumeration and Search*, CRC Press, (1999).

[36] E.M. Luks, Isomorphism of graphs of bounded valence can be tested in polynomial time, *J. Comput. System Sci.* **25** (1982), 42–65.

[37] B.D. McKay, `nauty` user's guide (version 1.5), Technical report TR-CS-90-02, Computer Science Department, Australian National University, 1990.

[38] B.D. McKay, Isomorph-free exhaustive generation, *J. Algorithms* **26** (1998), 306–324.

[39] B. D. McKay and N. C. Wormald, Uniform generation of random regular graphs of moderate degree, *J. Algorithms* **11** (1990), 52–67.

[40] P.M. Neumann, A lemma that is not Burnside's, *Math. Sci.* **4** (1979), 133–141.

[41] A. Nijenhuis and H. S. Wilf, *Combinatorial Algorithms, 2nd Edition*, Academic Press, (1978).

[42] W. Oberschelp, Kombinatorische anzahlbestimmungen in relationen, *Math. Ann.* **174** (1967), 53–58.

[43] R. Otter, The number of trees, *Ann. of Math.* **49** (1948), 583–599.

[44] C.H. Papadimitriou, *Computational Complexity*, Addison-Wesley, (1994).

[45] G.Pólya and R.C. Read, *Combinatorial Enumeration of Groups, Graphs, and Chemical Compounds*, Springer-Verlag, (1987).

[46] C.P. Schnorr, Optimal algorithms for self-reducible problems, in *Proceedings of the 3rd International Colloquium on Automata Theory, Languages and Programming* (1976), 322–337.

[47] L. Stockmeyer, The complexity of approximate counting (preliminary version), in *Proceedings of the 15th ACM Symposium on Theory of Computing* (1983), 118–126.

[48] G. Tinhofer, Generating graphs uniformly at random, *Computing, Supp.* **7** (1990), 235–255.

[49] S. Toda, PP is as hard as the polynomial-time hierarchy, *SIAM J. Comput.* **20** (1991), 865–877.

[50] L.G. Valiant, The complexity of enumeration and reliability problems, *SIAM J. Comput.* **8** (1979), 410–421.

[51] L.G. Valiant and V.V. Vazirani, NP is as easy as detecting unique solutions, *Theoret. Comput. Sci.* **47** (1986), 85–93.

[52] H. S. Wilf, The uniform selection of free trees, *J. Algorithms* **2** (1981), 204–207.

[53] N. C. Wormald, Some problems in the enumeration of labelled graphs, Ph.D. Thesis, Department of Mathematics, University of Newcastle, New South Wales, 1978.

[54] N. C. Wormald, The asymptotic connectivity of labelled regular graphs, *J. Combin. Theory Ser. B* **31** (1981), 156–167.

[55] N. C. Wormald, Generating random unlabelled graphs, *SIAM J. Comput.* **16** (1987), 717–727.

[56] S. Zhan, On Hamiltonian line graphs and connectivity, *Discrete Math.* **89** (1991), 89–95.

Department of Computer Science
University of Warwick
Coventry CV4 7AL
United Kingdom
leslie@dcs.warwick.ac.uk