Non Uniform Generation of Combinatorial Objects

Frank Weinberg and Markus E. Nebel

Fachbereich Informatik, Technische Universität Kaiserslautern

Abstract

In this article we present a method to generate random objects from a large variety of combinatorial classes according to a given distribution. Given a description of the combinatorial class and a set of sample data our method will provide an algorithm that generates objects of size n in worst-case runtime $\mathcal{O}(n^2)$ ($\mathcal{O}(n \cdot \log(n))$ can be achieved at the cost of a higher average-case runtime), with the generated objects following a distribution that closely matches the distribution of the sample data.

1 Introduction

When testing an algorithm one is often confronted with the problem of generating suitable input data. Since the possible inputs can usually be modeled as combinatorial classes the desire for procedures that generate the objects from these classes arises.

It is however usually neither feasible to test algorithms with all possible inputs nor does one know which of the possible inputs are the most interesting ones. Hence it is common practice to test with random input data. This immediately leads to the problem of generating random objects from a combinatorial class, usually of a preliminarily fixed size in order to get results which can be easily compared with each other.

Because of its importance this problem has been studied for quite some time and many results have been achieved both in terms of procedures for specific classes (see [3] or [15] for examples,) as well as more general approaches ([9], [10]).

A restriction common to all of these procedures is that they do not allow to specify a distribution among the objects of the considered combinatorial class, i.e. objects are generated according to a uniform distribution. This usually is no problem as long as only the correctness of the tested algorithm is considered. If, however, we want to do simulations to gather information about real-life behaviour of our algorithms (e.g. concerning runtime) we need to generate objects according to distributions observed on real-life data.

For those combinatorial classes which can be described by context-free grammars there exist algorithms to train the distribution of a given set of objects into the grammar and use the resulting stochastic context-free grammar to generate more data with a distribution similar to the one of the sample data. "However, they do not allow the user to fix the length of these sequences."([13])

In this paper we will present a method to combine the advantages of both approaches, that is to generate objects of a previously fixed length according to the distribution of a given set of sample data.

The general proceeding of our method is in two steps: First we equip the productions of a context-free grammar with probabilities such that the induced distribution on the generated language models the distribution of the sample data as closely as possible. Methods for doing so are well known; one of them is presented in Section 2.2.

After being trained this way, the (now weighted) context-free grammar is translated into an admissible specification. Admissible specifications, presented in Section 2.1 and extended for our purpose in Section 3.1, are a framework for specifying combinatorial classes, that immediately yields generating algorithms for the specified classes. The translation is presented in Sections 3.2 and 3.3.

2 Known Results and Basic Definitions

2.1 Random Generation

In order to generate objects from combinatorial classes we wil use *unranking algorithms*, that is, algorithms that, given k and n, generates the k-th object of size n from the class according to a previously fixed order. Obviously by drawing k at random this can be used to generate random objects.

We do this by extending an approach introduced by Flajolet, Zimmermann and Van Cutsem in [4] for the (uniform) random generation and adapted to the problem of unranking by Martínez and Molinero. Their results are collected in [10].

The basic principle of the approach is to decompose a combinatorial class into simpler classes (or, viewed the other way around, construct more complex classes from simpler ones) by using so-called admissible constructions. Formally:

Definition 2.1 ([5])**.** *Objects of size* 0 *are called* neutral objects *or* tags*, a class consisting of a single neutral object* ϵ *is called a* neutral class*. We will denote neutral classes by* $\mathcal{E}$ *(*$\mathcal{E}_\Box, \mathcal{E}_1, \mathcal{E}_2, \ldots$ *if we need to distinguish multiple neutral classes containing the objects* $\Box, \epsilon_1, \epsilon_2, \ldots$ *respectively).*

A class consisting of a single object of size 1 *is called an* atomic class*. We will denote atomic classes by* $\mathcal{Z}$ *(*$\mathcal{Z}_a, \mathcal{Z}_b, \ldots$ *to distinguish the classes containing the atoms* $a, b, \ldots$*).*

Definition 2.2 ([5])**.** *The* counting sequence *of a combinatorial class* $\mathcal{A}$ *is the sequence of integers* $(a_n)_{n\geq 0}$ *where* $a_n = card(\mathcal{A}^n)$ *is the number of objects in class* $\mathcal{A}$ *that have size (*$\widehat{=}$ *number of atoms)* n*.*

Definition 2.3 ([5])**.** *Assume that* Φ *is a construction that associates to a finite collection of classes* $\mathcal{B}, \mathcal{C}, \ldots$ *a new class*

$$\mathcal{A} := \Phi[\mathcal{B}, \mathcal{C}, \ldots],$$

in a finitary way: each $\mathcal{A}^n$ *depends on finitely many of the* $\{\mathcal{B}^i\}, \{\mathcal{C}^j\}, \ldots$ *Then* Φ *is* admissible *iff the counting sequence* (a_n) *of* $\mathcal{A}$ *only depends on the counting sequences* $(b_i), (c_j), \ldots$ *of* $\mathcal{B}, \mathcal{C}, \ldots$

In order to represent context-free grammars we will need two well-known admissible constructions.

Definition 2.4 ([5])**.** *If* $\mathcal{A}_1, \ldots, \mathcal{A}_k$ *are combinatorial classes and* $\epsilon_1, \ldots, \epsilon_k$ *are neutral objects, the* combinatorial sum *or* disjoint union $\mathcal{A}_1 + \ldots + \mathcal{A}_k$ *of* $\mathcal{A}_1, \ldots, \mathcal{A}_k$ *is defined as*

$$\mathcal{A}_1 + \ldots + \mathcal{A}_k := (\mathcal{E}_1 \times \mathcal{A}_1) \cup \ldots \cup (\mathcal{E}_k \times \mathcal{A}_k),$$

where $\cup$ *denotes set theoretic union.*

If $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *are combinatorial classes, the* cartesian product $\mathcal{A}_1 \times \mathcal{A}_2$ *of* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *is defined as*

$$\mathcal{A}_1 \times \mathcal{A}_2 := \{(\alpha_1, \alpha_2) | \alpha_i \in \mathcal{A}_i\}.$$

The size $|(\alpha_1, \alpha_2)|$ *is defined to be* $|\alpha_1| + |\alpha_2|$*.*

Now to be able to come up with unranking algorithms we need to specify an ordering on the objects of the same size that belong to the considered combinatorial class. We will do this recursively, following the structure of the class' admissible specification:

Definition 2.5 ([10])**.**

- *Neutral and atomic classes contain only one element, hence there is only one possible ordering.*
- *If* $\mathcal{B} = \mathcal{A}_1 + \ldots + \mathcal{A}_k$ *and* $\beta, \beta' \in \mathcal{B}^n$

$$\begin{aligned}\beta <_{\mathcal{B}^n} \beta' \Leftrightarrow & (\beta \in (\mathcal{A}_i)^n \text{ and } \beta' \in (\mathcal{A}_j)^n \text{ and } i < j) \text{ or} \\ & (\beta, \beta' \in (\mathcal{A}_i)^n \text{ and } \beta <_{(\mathcal{A}_i)^n} \beta').\end{aligned}$$

- *If $\mathcal{B} = \mathcal{A}_1 \times \mathcal{A}_2$ and $\beta = (\alpha_1, \alpha_2), \beta' = (\alpha'_1, \alpha'_2) \in \mathcal{B}^n$*

$$\begin{aligned}\beta <_{\mathcal{B}^n} \beta' \Leftrightarrow & |\alpha_1| < |\alpha'_1| \text{ or} \\ & (j = |\alpha_1| = |\alpha'_1| \text{ and } \alpha_1 <_{(\mathcal{A}_1)^j} \alpha'_1) \text{ or} \\ & (\alpha_1 = \alpha'_1 \text{ and } \alpha_2 <_{(\mathcal{A}_2)^{n-j}} \alpha'_2).\end{aligned}$$

The order used above for products is called *lexicographic order.* It leads to unranking algorithms with a worst-case time complexity in $\mathcal{O}(n^2)$, n the size of the objects to be generated. This worst-case complexity can be improved to $\mathcal{O}(n \cdot \log n)$ by using the *boustrophedonic order* at the cost of increasing average-case runtime. Details can be found in [10].

Now the actual unranking algorithms are quite straightforward:

For neutral and atomic classes we simply return the single object from the class, if the parameters of the call are correct ([10]).

Given a disjoint union we first determine the correct subclass by succesively adding up their sizes until this sum would exceed the given rank and then compute the rank among this class by subtracting the sizes of the precedent classes (Algorithm 1).

Algorithm 1 Unranking of disjoint unions ([10])

```
object function unrank($\mathcal{A}_1 + \ldots + \mathcal{A}_n$, $n$: size, $i$: rank)
  $c := 0$;
  $j := 1$;
  $d :=$ size($\mathcal{A}_j, n$);
  while $c + d \leq i$ do
    $c := c + d$;
    $j := j + 1$;
    $d :=$ size($\mathcal{A}_j, n$);
  end while
  return unrank($\mathcal{A}_j, n, i - c$);
```

Note 2.1. *We do not return the tags that would be added according to Definition 2.4, since this behaviour is usually more convenient in practice and if one actually needs the tags, they can easily be added by changing the specification accordingly.*

In cartesian products the objects are primarily ordered according to the sizes of their constituent objects, hence we first determine these sizes in the same way we determined the correct subclass in unions. Having found these sizes and the rank among all objects with the same sizes, the actual ranks in the subcalsses can easily be determined using *div* and *mod* (Algorithm 2).

The sizes required for the algorithms above can be computed using the following recursion:

$$\text{size}(\mathcal{A}, n) := \begin{cases} 1 & \mathcal{A} \text{ is atomic and } n = 1, \\ 0 & \mathcal{A} \text{ is atomic and } n \neq 1, \\ 1 & \mathcal{A} \text{ is neutral and } n = 0, \\ 0 & \mathcal{A} \text{ is neutral and } n \neq 0, \\ \sum_{i=1}^{k} \text{size}(\mathcal{B}_i, n) & \mathcal{A} = \mathcal{B}_1 + \ldots + \mathcal{B}_k, \\ \sum_{i=0}^{n} \text{size}(\mathcal{B}, i) \cdot \text{size}(\mathcal{C}, n - i) & \mathcal{A} = \mathcal{B} \times \mathcal{C}. \end{cases} \tag{1}$$

2.2 Weighted Context-Free Grammars

Combinatorial classes can be modeled as formal languages. We will use this analogy in Section 3 to transfer results on how to train weights for formal languages onto combinatorial classes. Hence we will introduce the required definitions and results.

We assume the reader has basic knowledge of the notions concerning context-free languages. An introduction can be found in [7].

Algorithm 2 Unranking of cartesian products ([10])

```
object function unrank($\mathcal{A} \times \mathcal{B}$, $n$: size, $i$: rank)
  $c := 0$;
  $j := 0$;
  $d := \text{size}(\mathcal{A}, j) \cdot \text{size}(\mathcal{B}, n - j)$;
  while $c + d \leq i$ do
    $c := c + d$;
    $j := j + 1$;
    $d := \text{size}(\mathcal{A}, j) \cdot \text{size}(\mathcal{B}, n - j)$;
  end while
  $i' := i - c$;
  $b := \text{size}(\mathcal{B}, n - j)$;
  $\alpha := \text{unrank}(\mathcal{A}, j, i'$ div $b)$;
  $\beta := \text{unrank}(\mathcal{B}, n - j, i'$ mod $b)$;
  return $(\alpha, \beta)$;
```

Definition 2.6 ([11])**.** *A* weighted context-free grammar (WCFG) *is a 5-tuple $G = (I, T, R, S, W)$, where I (resp. T) is an alphabet (finite set) of intermediate (resp. terminal) symbols (I and T are disjoint), $S \in I$ is called* axiom*, $R \subset I \times (I \cup T)^*$ is a finite set of production rules and $W : R \rightarrow \mathbb{R}^+$ is a mapping such that each rule $f \in R$ is equipped with a weight $w_f := W(f)$. In the sequel we will write $A \overset{w_f}{\rightarrow} \alpha$ instead of $f = (A, \alpha) \in R, w_f = W(f)$.*

If G is a WCFG, G is a stochastic context-free grammar (SCFG) *iff the additional restrictions*

$$\forall f \in R : W(f) \in (0, 1],$$
$$\forall A \in I : \sum_{f \in R, Q(f) = A} w_f = 1,$$

hold, where $Q(f)$ denotes the premise *of the production f, that is, the first component A of a production rule $(A, \alpha) \in R$. (We will call the second component the* conclusion *of (A, α).)*

Words are generated as for usual context-free grammars, the product of the weights of the used production rules gives the weight of a parse tree Δ, the sum of the weights of all possible full-parse trees of a word w gives the weight of w. We will write $W(\Delta)$ resp. $W(w)$ for these weights.

Definition 2.7. *If G, G' are WCFGs, G and G' are said to be* word-equivalent *iff $L(G) = L(G')$ and for each word $w \in L(G) : W(w) = W'(w)$. If additionally there exists a bijection f from the full-parse trees of w in G to the full-parse trees of w in G' such that for each tree Δ: $W(\Delta) = W'(f(\Delta))$ we will call G and G'* (derivation-)equivalent*.*

A WCFG G is called loop-free*, iff there exists no nonempty derivation $A \overset{+}{\Rightarrow} A, A \in I$. It is called* ϵ-free*, iff $\nexists (A, \epsilon) \in R, A \neq S$ and $\nexists (A, \alpha_1 S \alpha_2) \in R$, where ϵ denotes the empty word.*

The following result shows that given a CFG G and a set of full-parse trees to that grammar distributed according to some unknown distribution of $L(G)$ we can model the distribution by setting the weights on the grammar to relative frequencies with which the productions occur in the set of parse trees. This is based on the concept of maximum likelihood as introduced by Fisher in 1912-1922 ([1]).

Definition 2.8 ([14])**.** *For a given* treebank*, i.e., for a non-empty and finite corpus of full-parse trees, the* treebank grammar *$G = (I, T, R, S, W)$ is a SCFG defined by*

1. *I (resp. T; R) consist of exactly the intermediate symbols (resp. terminal symbols; productions) appearing in the treebank; S is the root of all the full-parse trees,*
2. *$\forall p = (A, \alpha) \in R : W(p) = \frac{f(r)}{\sum_{p' = (A, \alpha') \in P} f(p')}$,*

where $f(p)$ denotes the number of occurrences of p in the set of full-parse trees.

Theorem 2.1 ([14])**.** *Let f_t be a non-empty and finite corpus of full-parse trees and let G be the treebank grammar read off from f_t. Then the weights induced on the full-parse trees by G are a maximum-likelihood estimate of the probability model of the unweighted context-free grammar G' corresponding to G on f_t.*

3 Results

3.1 Weighting as an Admissible Construction

In order to encode the distribution into the combinatorial class we will equip the objects from our combinatorial class with weights.

For practical reasons we restrict ourselves to integral weights, modeling an object with weight λ as λ copies of the respective element. This can easily be achieved by introducing a new admissible construction:

Definition 3.1. *If $\mathcal{A}$ is a combinatorial class and λ is an integer, the* weighting *of $\mathcal{A}$ by λ, $\lambda\mathcal{A}$, is defined as*

$$\lambda\mathcal{A} := \underbrace{\mathcal{A} + \ldots + \mathcal{A}}_{\lambda \textit{ times}}.$$

We will call two objects from a combinatorial class copies of the same object *iff they only differ in the tags added by weighting operations.*

In order to unrank (resp. count) the objects of a class whose specification involves weighting, we could replace any weighting constructions by unions according to Definition 3.1, and use Algorithm 1 (resp. equation (1)). It is, however, especially if λ is large, more convenient to use a specialised algorithm, that takes into account that all the classes united are equal (Algorithm 3). For the size we find: $\mathrm{size}(\lambda\mathcal{A}, n) = \lambda \cdot \mathrm{size}(\mathcal{A}, n)$.

Algorithm 3 Unranking of weighted classes

```
object function unrank($\lambda\mathcal{A}$, $n$: size, $i$: rank)
  return unrank($\mathcal{A}, n, i$ mod size($\mathcal{A}, n$));
```

Since weighting a class can be replaced by a disjoint union the results from [10] concerning worst-case time complexity hold for classes whose specification involves weighting, meaning the unranking algorithms generated will have a runtime in $\mathcal{O}(n^2)$.

To conclude the introduction of this new construction we will have a look at the behaviour of weighted classes with respect to union and product as introduced in Section 2.1:

Lemma 3.1. *If $\mathcal{A}_1, \ldots, \mathcal{A}_n, \mathcal{B}$ are combinatorial classes, $\mathcal{B} = \mathcal{A}_1 + \ldots + \mathcal{A}_n$ and $\mathcal{A}_i$ contains λ copies of an object α, then $\mathcal{B}$ contains λ copies of (ϵ_i, α).*

If $\mathcal{A}_1, \ldots, \mathcal{A}_n, \mathcal{B}$ are combinatorial classes, $\mathcal{B} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_n$ and each $\mathcal{A}_i$ contains λ_i copies of an object α_i, then $\mathcal{B}$ contains $\prod_{i=1}^{n} \lambda_i$ copies of $\beta = (\alpha_1, \ldots, \alpha_n)$.

Proof. Immediate from Definition 2.4. □

3.2 Reweighting WCFGs

Since the weighing of classes allows only integral weights while the training as described in Section 2.2 introduces rational weights we have to reweight the grammars before we can transform them into admissible secifications.

As we will see in Section 3.3, the combinatorial objects we want to unrank in the end correspond to the full-parse trees in our grammar, objects of the same size corresponding to full-parse trees of words of the same length. Hence we have to make sure, that the relative weights of full-parse trees of words of the same length do not change, in order to guarantee, that the resulting unranking algorithms will produce the objects according to the distribution trained into the grammar.

If all productions are scaled by the same factor c, the weight of a derivation will scale by c^k, k being the number of steps in the derivation. Hence if all derivations of words of the same length involve the same number of steps, this operation will maintain the relative weights of these words. This is e.g. the case, if the grammar is in Chomsky normal form:

Theorem 3.2. *Let G be a WCFG in Chomsky normal form with all the weights appearing in G being rational; let c be the common denominator of the weights appearing in G, and G' be derived from G by multiplying all weights by c. Then all the weights in G' are integral and for $w_1, w_2 \in L(G)$, $|w_1| = |w_2|$ and Δ_i a full-parse tree of $w_i, i \in \{1,2\}$, in G and G'*

$$\frac{W'(\Delta_1)}{W'(\Delta_2)} = \frac{W(\Delta_1)}{W(\Delta_2)}$$

holds.

Proof. The first property follows immediately from the definition of c. If $|w_i| = 0, i \in \{1,2\}$, then $w_1 = w_2 = \epsilon$ and $\Delta_1 = \Delta_2$ consists of the single rule $S \to \epsilon$. Thus both fractions evaluate to 1 and the second property holds.
Now let $|w_i| \geq 1, i \in \{1,2\}$. Since G is in Chomsky normal form, Δ_i involves $|w_i| - 1$ applications of productions of the form $A \to BC, A, B, C \in I$, and $|w_i|$ applications of productions of the form $A \to a, A \in I, a \in T$. Thus $W'(\Delta_i) = W(\Delta_i) \cdot c^{2|w_i|-1}, i \in \{1,2\}$, and the second property follows immediately. □

If we are given a non-ambiguous SCFG, we can preprocess it in order to be able to use Theorem 3.2:

Theorem 3.3 ([8])**.** *If G is a SCFG, there exists a SCFG G' in Chomsky normal form that is word-equivalent to G, and G' can be effectively constructed from G.*

The construction given in [8] assumes that G is ϵ-free. It can however be extended to non-ϵ-free grammars by adding an additional step after the intermediate grammar G_1 has been created:

- For each production $A \overset{\lambda_1}{\to} \epsilon$ remove this production and for each instance of A in the conclusion of a production $B \overset{\lambda_2}{\to} \beta_1 A \beta_2$ add a new rule $B \overset{\lambda_1 \cdot \lambda_2}{\to} \beta_1 \beta_2$.
- If A occurs multiple times in a single conclusion, there has to be one new production for each combination of As removed, the weight being $\lambda_2 \cdot \lambda_1^l$ if l instances of A are deleted.

In principle the former two theorems already suffice for many practical purposes. However the transformation of Theorem 3.3 provides only word-equivalence while in some cases derivation-equivalence is needed. Hence we will present another method for reweighting.

Instead of scaling all productions by the same factor and ensuring that derivations are of equal length for words of the same size, we will scale the productions according to how much they contribute to the length of the word, that is, productions lengthening the word by k will be scaled by c^k. Since we consider context-free grammars, the lengthening of a production is given by "length of conclusion−1".

However, this rule leads to productions with a conclusion of length 1 not being reweighted, hence we have to assure, that all those productions already have integral weights in order to get a grammar with only integral weights after reweighting.

Additionally productions with a conclusion of length 0, i.e. ϵ-productions, would be rescaled by $\frac{1}{c}$ and thus they have to be avoided completely.

Obeying these restrictions would however make it impossible to generate words of length 0 or 1 with appropriate weights. Thus, to circumvent this problem, we will modify the required structure of the grammar a bit:

The axiom does not occur in any conclusion and any production having the axiom as premise has a conclusion of length 0 or 1. All other productions obey the restrictions stated above. This assures, that each derivation includes exactly one application of a production having the axiom as premise and, unless the word generated is the empty word, this application doesn't influence the length of the sentential form. Thus these productions can be reweighted seperately. Formally:

Definition 3.2. *If $G = (I, T, R, S, W)$ is a WCFG, G is said to be in* reweighting normal form (RNF) *iff*

1. *G is loop-free and ϵ-free;*
2. *$\forall A \to \alpha \in R, A = S : |\alpha| \leq 1$;*
3. *$\forall A \to \alpha \in R, A \neq S : |\alpha| > 1$ or $W(A \to \alpha) \in \mathbb{N}$;*
4. *$\forall A \in I \exists \alpha \in (I \cup T)^* : A \to \alpha \in R$.*

The last condition, that any intermediate symbol occurs as premise of at least one production, is not required for reweighting. It is, however, necessary for the translation of the grammar into an admissible specification later on.

Theorem 3.4. *Let G be a WCFG in RNF with all the weights appearing in G being rational; let s be the common denominator of the weights of productions with premise S in G, c be the common denominator of the weights of the remaining productions and G' be derived from G by multiplying the weights of productions with source S by s, and by multiplying the weights of productions $A \to \alpha, A \neq S$ by $c^{|\alpha|-1}$. Then all the weights in G' are integral and for $w_1, w_2 \in L(G)$, $|w_1| = |w_2|$ and Δ_i a full-parse tree of $w_i, i \in \{1, 2\}$, in G and G'*

$$\frac{W'(\Delta_1)}{W'(\Delta_2)} = \frac{W(\Delta_1)}{W(\Delta_2)}$$

holds.

Proof. Since $A \to \alpha \in R, A \neq S$ implies $|\alpha| > 1$ the first property follows immediately from the definition of s and c.
If $|w_i| = 0, i \in \{1, 2\}$, then $w_1 = w_2 = \epsilon$ and $\Delta_1 = \Delta_2$ consists of the single rule $S \to \epsilon$. Thus both fractions evaluate to 1 and the second property holds.
Now let $|w_i| \geq 1, i \in \{1, 2\}$. Since G is in RNF Δ_i involves 1 application of a production of the form $S \to \alpha, |\alpha| = 1$ and k applications of productions of the form $A_{i,j} \to \alpha_{i,j}, A_{i,j} \neq S$. Thus $\sum_{j=1}^{k} (|\alpha_{i,j}| - 1) = |w_i|, i \in \{1, 2\}$, and we find $W'(\Delta_i) = W(\Delta_i) \cdot s \cdot \prod_{j=1}^{n} c^{|\alpha_j|-1} = W(\Delta_i) \cdot s \cdot c^{|w_i|}, i \in \{1, 2\}$, and the second property follows immediately. □

Now, what remains to be done in this section is to transform arbitrary WCFGs to equivalent grammars in reweighting normal form. The proof of the following theorem describes how this can be done.

Theorem 3.5. *If G is a loop-free, ϵ-free WCFG, there exists a WCFG G' in RNF that is equivalent to G and G' can be effectively constructed from G.*

Proof. Let CH denote the set of all quadruples $(A, A_1 A_2 \ldots A_n, \lambda_0 \cdot \lambda_1 \cdot \ldots \cdot \lambda_n, B)$ with $A, A_1, \ldots, A_n \in I, B \in I \cup T, \lambda_i \in \mathbb{R}$ such that

$$A \xrightarrow{\lambda_0} A_1 \xrightarrow{\lambda_1} \ldots A_n \xrightarrow{\lambda_n} B$$

(for $n = 0$, we get $(A, \epsilon, \lambda, B) \in \text{CH}$ whenever $A \xrightarrow{\lambda} B$ is a rule with $B \in I \cup T$ and $n = -1$ results in $(A, \epsilon, 1, A) \in \text{CH}$ for $A \in N$). Since G is loop-free, CH is finite.

Now define G' as follows: G' contains the same terminals as G. The nonterminals are those from G, a new start symbol S' and the set CH. The rules are as follows:

- $(A, \gamma, \lambda, B) \xrightarrow{1} B$ for $(A, \gamma, \lambda, B) \in \text{CH}$,

- $A \xrightarrow{\lambda_0\lambda_1\cdots\lambda_n} w_0(A_1,\gamma_1,\lambda_1,B_1)w_1(A_2,\gamma_2,\lambda_2,B_2)\dots(A_n,\gamma_n,\lambda_n,B_n)w_n$
  for $A \xrightarrow{\lambda_0} \alpha = w_0A_1w_1A_2\dots A_nw_n$ a rule from G with $|\alpha| \geq 2$, $w_i \in T^*$ and $A_i \in I$,

- $S' \xrightarrow{\lambda} (S,\gamma,\lambda,a)$ for $(S,\gamma,\lambda,a) \in \mathrm{CH}$,

- $S' \xrightarrow{\lambda} \epsilon$, if $S \xrightarrow{\lambda} \epsilon$ is a rule in G.

From this construction it is obvious that G' satisfies conditions 1–3 of Definition 3.2. Condition 4 can be obtained by removing all nonterminals that violate it along with the productions containing them. Since these symbols and productions can not appear in any full-parse tree this removal does not influence derivation equivalence.

Since CH can easily be computed from G, G' can be constructed effectively.

It remains to be shown that G and G' are actually derivation-equivalent.

Claim. *For each full-parse tree $\Delta' : S \overset{*}{\Rightarrow} w, w \in T^*$ in G' there exists a full-parse tree $\Delta : S \overset{*}{\Rightarrow} w$ in G with $W(\Delta) = W'(\Delta')$.*

Proof. If $w = \epsilon$ then Δ' consists of the single rule $S' \xrightarrow{\lambda} \epsilon$ and Δ consisting of the single rule $S \xrightarrow{\lambda} \epsilon$ is a full-parse tree for w in G with $W(\Delta) = W'(\Delta')$.

If $|w| = 1$, Δ' consists of the rules $S' \xrightarrow{\lambda} (S,\gamma,\lambda,a)$ and $(A,\gamma,\lambda,a) \xrightarrow{1} a$ and Δ consisting of the rules that led to the inclusion of (S,γ,λ,a) in CH is a full-parse tree for w in G with $W(\Delta) = W'(\Delta')$.

Otherwise Δ' consists of one instance of the rule $S' \xrightarrow{1} S$ and several groups of rules $A \xrightarrow{\lambda_0\lambda_1\cdots\lambda_n} w_0(A_1,\gamma_1,\lambda_1,B_1)w_1(A_2,\gamma_2,\lambda_2,B_2)\dots(A_n,\gamma_n,\lambda_n,B_n)w_n$, $(A_i,\gamma_i,\lambda_i,B_i) \xrightarrow{1} B_i$. For each of these groups there is in G a production $A \xrightarrow{\lambda_0} \alpha = w_0A_1w_1A_2\dots A_nw_n$ and if $A_i \neq B_i$ the set of productions that led to the inclusion of $(A_i,\gamma_i,\lambda_i,B_i)$ in CH. Additionally by construction of CH and G' the product of weights is the same for both groups. Thus by combining the groups of productions from G derived from the groups of productions in Δ' we get a full-parse tree for w in G with $W(\Delta) = W'(\Delta')$. □

Claim. *For each full-parse tree $\Delta : S \overset{*}{\Rightarrow} w, w \in T^*$ in G there exists a full-parse tree $\Delta' : S \overset{*}{\Rightarrow} w$ in G' with $W(\Delta) = W'(\Delta')$.*

Proof. By applying the argumentation from the above proof in reverse. □

Together the proofs of the two claims give the bijection required for derivation-equivalence. □

Neither of the requirements in Theorem 3.5 may be dropped without losing derivation-equivalence:

- Concerning ϵ-freeness consider the set of productions $\{S \to \epsilon, S \to A, A \to \epsilon\}$ where $\epsilon \in L(G)$ has two different full-parse trees while in an ϵ-free grammar ϵ can only have the single derivation $S \to \epsilon$.

- Concerning loop-freeness note that if $A \overset{+}{\Rightarrow} A$ is a loop in G and A appears in a derivation of $w \in L(G)$, w has infinitely many full-parse trees in G while in a loop-free grammar any word w has only finitely many parse trees.

To transform a grammar to a word-equivalent loop-free, ϵ-free grammar, the ideas used in the proof of Theorem 3.3 can be applied. Note, however, that eliminating loops with a weight ≥ 1, will result in infinite weights.

3.3 Transforming WCFGs into Admissible Specifications

To get the transformation from WCFGs to admissible specifications, we note how the elements of grammars can be modeled by elements of specifications:

- The empty word is modeled as neutral class.
- Terminal symbols are modeled as atomic classes, containing the respective symbol.
- Intermediate symbols are modeled as classes that contain all words that can be derived from the symbol.
- Productions are modeled as classes containing the cartesian product of all symbols of the conclusion.
- Weights of productions are modeled by weighting the respective classes.
- Multiple productions with the same source are modeled as disjoint union of the classes modeling the (weighted) productions.

Formalizing this, we get:

Theorem 3.6. *Let $G = (I, T, R, S, W)$ be a WCFG with integral weights. For each $A \in I$ let $\{A \overset{\lambda_{A,j}}{\to} \alpha_{A,j,1} \dots \alpha_{A,j,n_{A,j}} | 1 \le j \le n_A\}$ denote the complete set of productions with premise A, n_A being the number of such productions. Let $\mathcal{T}$ be constructed as follows:*

- *For each $a \in T$ add an atomic class $\mathcal{Z}_a$.*
- *For each $A \in I$ add a class $\mathcal{A}$.*
- *For each $A \overset{\lambda_{A,j}}{\to} \alpha_{A,j,1} \dots \alpha_{A,j,n_{A,j}} \in G, 1 \le j \le n_A$, add a class $\mathcal{A}_j$ and an equation $\mathcal{A}_j = \beta_{A,j,1} \times \dots \times \beta_{A,j,n_{A,j}}$ where $\beta_{A,j,k} = \mathcal{Z}_a$ if $\alpha_{A,j,k} = a \in T$ and $\beta_{A,j,k} = \mathcal{B}$ if $\alpha_{A,j,k} = B \in I$.*
  If $n_{A,j} = 0$ add the equation $\mathcal{A}_j = \mathcal{E}$ instead of $\mathcal{A}_j = \beta_{A,j,1} \times \dots \times \beta_{A,j,n_{A,j}}$.
- *For each $A \in I$ add an equation $\mathcal{A} = \lambda_{A,1}\mathcal{A}_{A,1} + \dots + \lambda_{A,n_A}\mathcal{A}_{A,n_A}$.*

Then $\mathcal{T}$ is an admissible specification and for each $w \in L(G)$ there exists a bijection f between the full-parse trees Δ of w in G and those objects in $\mathcal{S}_{|w|}$ that contain the atomic classes $\mathcal{Z}_{a_1} \dots \mathcal{Z}_{a_{|w|}}$ in the same order as the terminal symbols a_i appear in w such that $\mathcal{S}_{|w|}$ contains $W(\Delta)$ copies of $f(\Delta)$.

Proof. The first property follows immediately from the construction of $\mathcal{T}$.
Considering the second property we will first show the following lemma:

Lemma 3.7. *Let $\mathcal{T}$ be constructed as in Theorem 3.6, $\Delta = A \overset{*}{\Rightarrow} w$, $w \in L((I, T, R, A, W))$, $\lambda = W(\Delta)$. Then there are λ copies of an object $o \in \mathcal{A}^{|w|}$ that contains the atomic classes $\mathcal{Z}_{a_1} \dots \mathcal{Z}_{a_{|w|}}$ in the same order as the terminal symbols a_i appear in w.*

Proof. Let n denote the number of steps in the derivation of w, $A \overset{\mu}{\to} \alpha_1 \dots \alpha_k$ be the first production used in Δ, $\mathcal{B}$ be the class introduced for this production and ϵ_B be the tag associated with $\mathcal{B}$ in the equation for $\mathcal{A}$.

$n = 1$: Then $\lambda = \mu$ and $w = \alpha_1 \dots \alpha_k$ and by construction of $\mathcal{T}$:
$(\alpha_1, \dots, \alpha_k) \in \mathcal{B}$ and thus $\forall 1 \le i \le \lambda : (\epsilon_B, (\epsilon_i, (\alpha_1, \dots, \alpha_k))) \in \mathcal{A}$ where the ϵ_i are introduced during the weighting operation $\lambda\mathcal{B}$. Hence the Lemma holds.

$n > 1$: Assume the Lemma holds for all derivations of lengths $< n$.
The derivations $\alpha_i \overset{*}{\Rightarrow} w_i$, where $w_i \in T^*$, $w = w_1 \dots w_k$, all have lengths $< n$ hence if $\alpha_i \in I$ we may set o_i to the object corresponding to the derivation $\alpha_i \overset{*}{\Rightarrow} w_i$. If $\alpha_i \in T$ we set $o_i = \alpha_i$. Then by construction of $\mathcal{T}$:
$(o_1, \dots, o_k) \in \mathcal{B}$ and thus $\forall 1 \leq i \leq \lambda : (\epsilon_B, (\epsilon_i(o_1, \dots, o_k))) \in \mathcal{A}$ where the ϵ_i are introduced during the weighting operation $\lambda\mathcal{B}$. Using the induction hypothesis and Lemma 3.1 the Lemma follows.

□

It remains to be shown that conversely each object $o \in \mathcal{S}$ models a derivation:
From the construction of $\mathcal{T}$, we get that each $o \in \mathcal{A}$, $\mathcal{A}$ added for $A \in I$, must be of the form $o = (\epsilon_j, (\epsilon_l, (o_1, \dots, o_k)))$, where each of the $o_i, 1 \leq i \leq k$, is either a terminal symbol or $o_i \in \mathcal{B}_i$, $\mathcal{B}_i$ added for $B_i \in I$ and o_i is smaller than o. Hence assuming that for smaller o_i there exist corresponding derivations, we get a derivation corresponding to o by concatenating $A \rightarrow \alpha_1 \dots \alpha_k$, $\alpha_i = o_i$ if $o_i \in T$, $\alpha_i = B_i$ if $o_i \in \mathcal{B}_i$ with the derivations for those o_i not in T. The property follows by induction. □

It is easy to verify that the above transformation can also be applied inversely to get a CFG from an admissible specification involving only disjoint unions and cartesian products. Additionally the admissible construction of sequence generation can also be transformed into productions of a CFG, translating $R = SEQ(S)$ into $R \mapsto \epsilon | RS$.

4 Example of Application

As a simple example we have used our method to generate random RNA secondary structures. These are typically modelled as correctly parenthesized words over the alphabet $\{[, |,]\}$, where corresponding brackets represent bases paired with each other and | denotes an unpaired base. (A more thorough introduction to the topic can be found in [2].)

Using a simple grammar for these words and training it using the data from [16] we arrived at the following:

$$\begin{array}{rlrlrl}
1: & S \rightarrow R, & 0.31: & R \rightarrow T, & 0.69: & T \rightarrow |, \\
 & & 0.69: & R \rightarrow TR, & 0.31: & T \rightarrow [R].
\end{array}$$

Since this grammar is loop-free and ϵ-free, we can use the procedure from Theorem 3.5 to transform it to RNF.

We find CH $= \{(S, \epsilon, 1, S), (S, \epsilon, 1, R), (S, R, 0.31, T), (S, RT, 0.21, |), (R, \epsilon, 1, R), (R, \epsilon, 0.31, T), (R, T, 0.21, |), (T, \epsilon, 1, T), (T, \epsilon, 0.69, |)\}$ and thus after reducing get the new grammar

$$\begin{array}{rlrl}
1: & S' \rightarrow (S, \epsilon, 1, R), & 0.69: & R \rightarrow (T, \epsilon, 1, T)(R, \epsilon, 1, R), \\
0.31: & S' \rightarrow (S, R, 0.31, T), & 0.21: & R \rightarrow (T, \epsilon, 1, T)(R, \epsilon, 0.31, T), \\
0.31: & S' \rightarrow (S, RT, 0.31, |), & 0.14: & R \rightarrow (T, \epsilon, 1, T)(R, T, 0.21, |), \\
0.31: & T \rightarrow [(R, \epsilon, 1, R)], & 0.48: & R \rightarrow (T, \epsilon, 0.69, |)(R, \epsilon, 1, R), \\
0.10: & T \rightarrow [(R, \epsilon, 0.31, T)], & 0.15: & R \rightarrow (T, \epsilon, 0.69, |)(R, \epsilon, 0.31, T), \\
0.07: & T \rightarrow [(R, T, 0.21, |)], & 0.10: & R \rightarrow (T, \epsilon, 0.69, |)(R, T, 0.21, |), \\
1: & (A, \gamma, \lambda, B) \rightarrow B, \quad (A, \gamma, \lambda, B) \in \text{CH}, \quad B \neq S. & &
\end{array}$$

Since replacing each of the (A, γ, λ, B) by B in the conclusions does not lead to two of the rules becoming identical, this transformation is derivation equivalent for our grammar and gives the more readable version

$$\begin{array}{rlrlrlrl}
1: & S' \rightarrow R, & 0.69: & R \rightarrow TR, & 0.48: & R \rightarrow |R, & 0.31: & T \rightarrow [R], \\
0.31: & S' \rightarrow T, & 0.21: & R \rightarrow TT, & 0.15: & R \rightarrow |T, & 0.10: & T \rightarrow [T], \\
0.31: & S' \rightarrow |, & 0.14: & R \rightarrow T|, & 0.10: & R \rightarrow ||, & 0.07: & T \rightarrow [|].
\end{array}$$

Applying Theorem 3.4 we find the smallest common denominators to be $s = 100$ and $c = 100$ and thus the reweighted set of productions is:

$$\begin{array}{rlrlrlrl}
1: & S' \to R, & 69: & R \to TR, & 48: & R \to |R, & 3100: & T \to [R], \\
31: & S' \to T, & 21: & R \to TT, & 15: & R \to |T, & 1000: & T \to [T], \\
31: & S' \to |, & 14: & R \to T|, & 10: & R \to ||, & 700: & T \to [|].
\end{array}$$

Applying Theorem 3.6, we get the following specification:

$$\begin{array}{lcllcllcllcl}
\mathcal{S}_1 & = & \mathcal{R}, & \mathcal{R}_1 & = & \mathcal{T} \times \mathcal{R}, & \mathcal{R}_4 & = & \mathcal{Z}_{|} \times \mathcal{R}, & \mathcal{T}_1 & = & \mathcal{Z}_{[} \times \mathcal{R} \times \mathcal{Z}_{]}, \\
\mathcal{S}_2 & = & \mathcal{T}, & \mathcal{R}_2 & = & \mathcal{T} \times \mathcal{T}, & \mathcal{R}_5 & = & \mathcal{Z}_{|} \times \mathcal{T}, & \mathcal{T}_2 & = & \mathcal{Z}_{[} \times \mathcal{T} \times \mathcal{Z}_{]}, \\
\mathcal{S}_3 & = & \mathcal{Z}_{|}, & \mathcal{R}_3 & = & \mathcal{T} \times \mathcal{Z}_{|}, & \mathcal{R}_6 & = & \mathcal{Z}_{|} \times \mathcal{Z}_{|}, & \mathcal{T}_3 & = & \mathcal{Z}_{[} \times \mathcal{Z}_{|} \times \mathcal{Z}_{]}, \\
\mathcal{S} & = & \multicolumn{10}{l}{100\mathcal{S}_1 + 31\mathcal{S}_2 + 21\mathcal{S}_3,} \\
\mathcal{R} & = & \multicolumn{10}{l}{69\mathcal{R}_1 + 21\mathcal{R}_2 + 14\mathcal{R}_3 + 48\mathcal{R}_4 + 15\mathcal{R}_5 + 10\mathcal{R}_6,} \\
\mathcal{T} & = & \multicolumn{10}{l}{3100\mathcal{T}_1 + 1000\mathcal{T}_2 + 700\mathcal{T}_3.}
\end{array}$$

We have implemented the resulting algorithm as well as an algorithm for uniform generation of RNA secondary structures in the computer algebra system Maple and used these implementations to generate 1000 objects of size 150 with each of the algorithms. The resulting ratios of unpaired bases are shown in Table 1.

Structures	Ratio of unpaired bases
Generated unweighted	45,1%
Genrated weighted	53,2%
From database	52,2%

Table 1: Ratio of unpaired bases for the generated structures.

As was to be expected, the structures generated with the weighted algorithm closely match the structures in the database concerning this parameter.

Concerning other parameters, such as the free energy of the molecules, the results generated by our weighted unranking algorithm aren't as good. This is due to the fact, that the grammar we started with is too simple (i.e. it has too few free variables) to model these parameters and hence there is no way to train their properties into this grammar.

This problem can however be circumvented by using a more sophisticated grammar. In [12] such a grammar is given for RNA secondary structures and it is shown that applying our approach to this grammar yields an algorithm that generates secondary structures with free energies distributed very similar to the distribution observed for real RNA molecules.

5 Concluding Remarks

In this paper we have shown, how to get algorithms for the random generation of combinatorial objects of a fixed size according to an arbitrary distribution, given a context-free grammar that models the class or an admissible specification of the class that can be transformed into such a grammar and given the distribution either as weights on the grammar or as a set of sample objects, distributed accordingly.

We have done so by extending the popular framework of admissible specifications of combinatorial classes to also include the specification of weights, allowing for the generic creation of algorithms that generate objects from combinatorial classes according to a non-uniform distribution and by furthermore showing, how weighted context-free grammars can be transformed into admissible specifications, hence allowing to use the well-known procedures for training the weights on context-free grammars to be used for the training of weights in admissible specifications.

However, we had to restrict ourselves to use only integral weights. Hence the weights returned by the training algorithms had to be rescaled, which in turn required a rather complex transformation of the grammar.

This transformation is however, as well as all other steps involved, easily automateable. Thus it appears to be a sensible task, to develop a tool that, given a specification or grammar and a set of sample objects, generates the corresponding unranking algorithm.

The algorithms presented generate objects of size n with a worst-case runtime in $\mathcal{O}(n^2)$. In their work [6], Flajolet, Fusy and Pivoteau show how algorithms for the uniform random generation of combinatorial objects that have a worst-case runtime linear in the size of the object generated can be automatically created from admissible specifications if one weakens the restriction for the algorithms to create objects of exact size n to the creation of objects with expected size n.

We believe it to be possible to combine their results with the results presented here in order to get algorithms for the weighted generation that yield a runtime linear in the size of the created objects. It remains to be seen, however, how exactly this can be achieved.

References

[1] John Aldrich. R. A. Fisher and the Making of Maximum Likelihood 1912-1922. *Statistical Science*, 12(3):162–176, 1997.

[2] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis*. Cambridge University Press, 1998.

[3] M. C. Er. Enumerating ordered trees lexicographically. *The Computer Journal*, 28(5):538–542, 1985.

[4] P. Flajolet, P. Zimmermann, and B. V. Cutsem. A calculus for the random generation of combinatorial structures. *Theoretical Computer Science*, 132(1-2):1–35, 1994.

[5] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.

[6] Philippe Flajolet, Éric Fusy, and Carine Pivoteau. Boltzmann sampling of unlabelled structures. http://algo.inria.fr/flajolet/Publications/FlFuPi06.pdf, 2006. Extended abstract.

[7] Michael A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, 1978.

[8] T. Huang and K.S. Fu. On stochastic context-free languages. *Inf. Sc.*, 3:201–224, 1971.

[9] Jens Liebehenschel. *Lexikographische Generierung, Ranking und Unranking kombinatorischer Objekte: Eine Average-Case Analyse*. PhD thesis, Johann Wolfgang Goethe-Universität Frankfurt am Main, 2000.

[10] Xavier Molinero. *Ordered Generation of Classes of Combinatorial Structures*. PhD thesis, Universitat Politècnica de Catalunya, 2005.

[11] Markus E. Nebel. On a Statistical Filter for RNA Secondary Structures. Technical report, Frankfurter Informatik-Berichte, 5 2002.

[12] Markus E. Nebel and Anika Scheid. Random generation of rna secondary structures according to native distributions. *Submitted*, 2010.

[13] Yann Ponty, Michel Termier, and Alain Denise. GenRGenS: software for generating random genomic sequences and structures. *Bioinformatics*, 22(12):1534–1535, 2006.

[14] Detlef Prescher. A tutorial on the expectation-maximization algorithm including maximum-likelihood estimation and em training of probabilistic context-free grammars. http://staff.science.uva.nl/ prescher/papers/bib/2003em.prescher.pdf, 2003.

[15] F. Ruskey. Generating t-ary trees lexicographically. *SIAM J. Comp.*, 7(4):424–439, 1978.

[16] J. Wuyts, P. De Rijk, Y. Van de Peer, T. Winkelmans, and R. De Wachter. The european large subunit ribosomal rna database. *Nucleic Acids Res.*, 29:175–177, 2001.