ELSEVIER

Information Processing Letters 49 (1994) 95–99

Information Processing Letters

Generating words in a context-free language uniformly at random

Harry G. Mairson

Department of Computer Science, Brandeis University, Waltham, MA 02254, USA

(Communicated by L.A. Hemaspaandra)
(Received 21 September 1992)
(Revised 7 October 1993)

Abstract

Given an unambiguous context-free grammar **G** and an integer $n > 0$, we give two algorithms for generating words derived from the grammar uniformly at random (u.a.r.). The first algorithm generates a word u.a.r. in $O(n^2)$ time using a data structure of size $O(n)$. The second algorithm generates a word u.a.r. in $O(n)$ time using a data structure of size $O(n^2)$. Both algorithms make use of a preprocessing stage, where a data structure is built to facilitate the later generation of words uniformly at random. The algorithms are analyzed in a unit-cost model where the grammar is considered as a constant, and a weighted probabilistic choice among m outcomes is computed in $O(m)$ time. When the unit-cost memory assumption is replaced by a logarithmic-cost measure, we give a simple approximation scheme that reduces the bit complexity of the above space bounds by a factor of $\Theta(\log n/n)$, while generating words with probability $N^{-1}(1 + O(n^{-k}))$ for any integer $k > 0$, where N is the number of derivable words of length n.

Key words: Algorithms; Analysis of algorithms

1. Introduction

Let **G** be an unambiguous context-free grammar generating some language L, and let L_n be the words in L of length n. We consider the problem of generating *uniformly at random* (u.a.r.) a word $w \in L_n$, i.e., where each such word w is considered equally likely to be generated. This problem was considered by Cohen and Hickey in [4], where they presented two algorithms: the first generates w in time $O(n^2(\log n)^2)$ time and requires $O(n)$ space, while the second generates w in time $O(n)$, and requires a data structure of size $O(n^{r+1})$, where r is the number of nonterminals in **G**.

In this paper, we improve these time and space bounds. We give a simple method that generates u.a.r. a word $w \in L_n$ in $O(n^2)$ time, using a data structure of size $O(n)$. A second algorithm generates w in $O(n)$ time, using a data structure of size $O(n^2)$.

As in [1,4], a data structure is first built depending on n and **G** in a *preprocessing phase*. This data structure is used in a later *generation phase*, where words of length n are produced uniformly at random. The space bounds given refer to the size of the data structure and to space overhead during the generation phase; the time bounds refer uniquely to the generation phase.

The model of computation assumes the unit-cost criterion, where the grammar is considered as a constant; time and space bounds are in terms of the length of the word generated. A probabilistic choice among m outcomes is assumed to be computed in $O(m)$ time. When the unit-cost assumption is replaced by a logarithmic-cost measure, we give a simple approximation scheme that reduces the bit complexity of the above space bounds by a factor of $\Theta(\log n/n)$, while generating words with probability $|L_n|^{-1}(1+O(n^{-k}))$ for any integer $k > 0$.

2. Preliminaries

We make computational use of an obvious relation between derivations and generating functions. In particular, given a context-free grammar $\mathbf{G}'$, we may associate every nonterminal V with a formal power series $\Sigma_n v_n z^n$, where v_n is an integer denoting the number of words of length n derivable from V. The most important consequence of this association is the so-called *convolution theorem*: if A and B are nonterminals, the number of words of length n that can be generated from the intermediate form AB is the coefficient of z^n in the product of the associated generating functions $A(z) = \Sigma_n a_n z^n$ and $B(z) = \Sigma_n b_n z^n$. The explanation is not at all mysterious, since the coefficient of interest is the convolution $\Sigma_{0<k<n} a_k b_{n-k}$, which counts all possible u of length k derived from A, concatenated with all possible v of length $n-k$ derived from B. Two important properties of the convolution are that it is commutative and associative.

For a canonical example of the use of the convolution theorem, consider words generated from the parenthesis grammar $P \to \varepsilon\,|(P)P$. From the nonterminal P, we derive a generating function $P(z) = \Sigma_n p_n z^n$, where p_n is the number of parenthesis strings of length n. The productions translate nicely into the equation $P(z) = 1 + zP(z)zP(z) = 1 + z^2P^2(z)$; the associated quadratic equation $P^2z^2 - P + 1 = 0$ is solved for P and expanded in a power series in z (see e.g., [2]). In the case of context-free grammars with more than one nonterminal, for example a generator for words representing trees (see [3]),

$$T \to xF$$
$$F \to \varepsilon|dTuF$$

(here T, F, d, u correspond to "tree", "forest", "down", and "up"), we translate to generating functions

$$T(z) = zF(z),$$
$$F(z) = 1 + z^2T(z)F(z),$$

derive a quadratic equation solely in $F(z)$ and solve it, substituting this solution back into $T(z)$.

While one can find simple, analytic, closed-form solutions for these examples, in general this is not always the case. An alternative is to simply compute the coefficients of z^i, $0 \leqslant i \leqslant n$, for all of the generating functions associated with nonterminals of the grammar by using dynamic programming. The number of words of length m derivable from some intermediate form $U_1 \ldots U_t$ can be calculated via dynamic programming by convolving the lower-order coefficients of the generating functions $U_i(z)$ associated with the nonterminals U_i.

3. The basic algorithm

Assume that the grammar $\mathbf{G}$ is given in Chomsky normal form without ε-productions except for the start symbol, so that all productions are of the form $A \to BC$ or $A \to a$. For each nonterminal A, define $\|A\|_l = |\{w\colon A \Rightarrow w \text{ and } |w| = l\}|$. In the preprocessing stage, the values $\|A\|_l$ for all A and $0 \leqslant l \leqslant n$ can be tabulated in $O(n)$ time and space using dynamic programming, where $A[l]$ is used to store $\|A\|_l$.

```
**for each** nonterminal $A$ **do** $A[1] := 0$;
**for each** production $A \to a$ **do**
    $A[1] := A[1] + 1$;
**for** $l \leftarrow 2$ **up to** $n$ **do**
    **for each** nonterminal $A$ **do**
        **for each** production $A \to BC$ **do**
            $A[l] := A[l] + \Sigma_{0<k<l} B[k] \cdot C[l-k]$
```

For a production $A \to BC$, define $\|A \to BC\|_l = \Sigma_{0<k<l} \|B\|_k \cdot \|C\|_{l-k}$, counting the words of length l generated by any derivation beginning with $A \to BC$. In the *generation phase*, to com-

pute a word a of length l from nonterminal A, choose a production $A \to BC$ with probability $\|A \to BC\|_l / \|A\|_l$, and a *split* $0 < k < l$ chosen with probability $\|B\|_k \cdot \|C\|_{l-k} / \|A \to BC\|_l$. Then recursively generate words b and c, where $a = bc$, $|b| = k$, $|c| = l - k$, b is generated u.a.r. from B, and C is generated u.a.r. from C.

Each recursive call chooses a production with unit cost, and a split with linear cost. The time taken to generate, from the start symbol, a word of length n chosen u.a.r. is then bounded as

$$T(n) \leq cn + \max_{1 \leqslant k < n} [T(k) + T(n-k)],$$

which has solution $T(n) = O(n^2)$. Observe if $k = 1$ repeatedly we may have n iterations, each taking $O(n)$ time.

4. Linear time generation

We now show how to generate words from $\mathbf{G}$ in linear time, given a quadratic data structure constructed in the preprocessing stage. We use a grammar $\mathbf{G}'$, constructed by replacing each nonterminal A in $\mathbf{G}$ by a set A_l of nonterminals, $1 \leqslant l \leqslant n$, and each production $A \to BC$ in $\mathbf{G}$ by a set of productions $A_l \to B_k C_{l-k}$, where $1 \leqslant l \leqslant n$ and $0 < k < l$. Clearly $A_l \Rightarrow w$ in $\mathbf{G}'$ iff $A \Rightarrow w$ in $\mathbf{G}$ and $|w| = l$.

In the preprocessing stage, we construct $\mathbf{G}'$, as well as a binary tree $T_l[A \to BC]$ for each production $A \to BC$ in $\mathbf{G}$ and $1 \leqslant l \leqslant n$. The purpose of such a tree is to optimize the choice of a split, which was the bottleneck in the previous generation algorithm.

The leaves of $T_l[A \to BC]$ contain a candidate split $0 < k < l$ and a weight $\|B\|_k \cdot \|C\|_{l-k}$. Define the weight of an internal node of $T_l[A \to BC]$ as the sum of the weights of leaves in the subtree rooted at that node.

To generate a word of length l from A beginning with the production $A \to BC$, traverse a path in $T_l[A \to BC]$ guided by the weights: beginning at the root with weight p, and left and right children with weights q and $p - q$, choose the left child with probability q/p, and the right child with probability $1 - q/p$. The cost of randomly constructing the path to a leaf, which we also refer to as a *tree walk*, is simply by average path length of the tree, as defined by the weights.

To construct the required tree, we use the following simple lemma, which is a variant of Huffman coding that facilitates our analysis:

Lemma 4.1. *Given M distinct keys $K = \{k_1, \ldots, k_M\}$ and associated frequencies $P = \{p_1, \ldots, p_M\}$, each $1/p_i$ a power of 2, and $\Sigma p_i \leqslant 1$, we can construct a binary trie for the keys where the path length from the root to k_i is bounded by $d_i = \log(1/p_i)$.*

Proof. We proceed by induction on M, coding the path from the root to each k_i by a string $\sigma_i \in \{0, 1\}^*$.

In the basis $M = 1$, the construction is trivial: set $\sigma_1 = \varepsilon$. Clearly $d_i \geqslant 0$ and the path length is 0.

When $M > 1$, consider the case that each p_i is distinct, so that no $p_i = p_j$ for $i \neq j$. If d_m is maximal among the d_i, set $\sigma_m = 0^{d_m}$, and for $i \neq m$, set $\sigma^i = 0^{d_i - 1}1$.

When $M > 1$ and $p_i = p_j$ for some $i \neq j$, construct by induction a trie T' over the $M - 1$ keys $K' = K - \{k_i, k_j\} \cup \{k'\}$, where key k' has probability $2p_i = p_i + p_j$. Let σ' code the path from the root to k' in T', with length at most $\log(1/2p_i) = d_i - 1$. To construct T, extend σ' to $\sigma_i = \sigma'0$ (for k_i) and $\sigma_j = \sigma'1$ (for k_j). □

Corollary 4.2. *Given M distinct keys $K = \{k_1, \ldots, k_M\}$ and associated probabilities $P = \{q_1, \ldots, q_M\}$, $\Sigma q_i \leqslant 1$, we can construct a binary tree for the keys where the path length from the root to k_i is bounded by $d_i = 1 + \log(1/q_i)$.*

Proof. For each q_i, let d_i be a solution to $q_i/2 < (\frac{1}{2})^{d_i} \leqslant q_i$, and set $p_i = (\frac{1}{2})^{d_i}$. Clearly $1/p_i$ is a power of 2, $\Sigma p_i \leqslant 1$, and $d_i \leqslant 1 + \log(1/q_i)$.

Using the above lemma, construct a trie over the keys K with probabilities $\{p_1, \ldots, p_M\}$. Then compress the trie into a binary tree, by removing internal nodes with only one child. □

Using the construction of the above corollary, tree $T_l[A \to BC]$ has the following property: a path from the root to the leaf marked with production $A_l \rightarrow B_k C_{l-k}$ is no more than

$$1 + \log(\| A \|_l / \| B \|_k \| C \|_{l-k}).$$

Theorem 4.3. *The tree construction allows generation of words from nonterminal A of length n uniformly at random with a time bound of* $O(n)$.

Proof. The choice of *productions* is clearly effected in $O(n)$ time, since the parse tree of the word generated is binary, and each of the $n-1$ internal nodes in the tree is associated uniquely with the choice of a production. To complete the proof, we show by induction on n that at most $n - 1 + \log \| A \|_n$ steps are needed to tree walk all relevant trees constructed using the method of the lemma and corollary.

We may assume by trivially rewriting the grammar that there are no productions $A \rightarrow a$ *and* $A \rightarrow b$ when A is not the start symbol. The basis of the induction is then when $n = 1$: since $A_1 \rightarrow a$ is the only possible production (for some terminal a), no tree walk is necessary, and $1 - 1 + \log \| A \|_1 = 0$.

For the inductive step, we assume by inductive hypothesis that the generation of B_k and C_{n-k} take no more than $k - 1 + \log \| B \|_k$ and $n - k - 1 + \log \| C \|_{n-k}$ tree steps, then the total number of tree steps in expanding A_n will be no more than

$$\begin{aligned} &1 + (\log \| A \|_n / (\| B \|_k \| C \|_{n-k})) \\ &\quad + (k - 1 + \log \| B \|_k) \\ &\quad + (n - k - 1 + \log \| C \|_{n-k}), \end{aligned}$$

which equals $n - 1 + \log \| A \|_n$ regardless of the choice of k.

Instead of trying to bound the cost of each "tree walk" that chooses a production and a split, we construct our data structure and devise our analysis considering the entire sequence of tree walks, using the properties of the constructed trees to amortize the total cost in an efficient manner. Notice that if a very long path is followed in to find a production and split $A_n \rightarrow B_k C_{n-k}$ to use, the positive compensation is that the number of subsequent possible derivations is reduced greatly, speeding up the computation later on. Since there is some constant number c of terminals in the language, we know $\| A \|_n \leqslant c^n$, and hence the cost of generating a word from A_n is $O(n)$.

5. An approximation method

While the linear-time algorithm is clearly time-optimal, it would be of interest to reduce its $O(n^2)$ space requirement. Even this space bound is deceptive: memory cells store the number of derivations from particular sums of nonterminals, and there are in the worst case $\Theta(c^n)$ derivations from any A_n. As a consequence, in the worst case $\Theta(n^3)$ bits may be required to represent these coefficients. We conclude by mentioning a simple space-efficient approximation method where numbers of derivations are stored in floating point, i.e., to k significant binary digits, and the remaining bits are truncated to zero.

Consider the generation described in the previous section, where the algorithm proceeds by traversing paths in trees directed stochastically by weights stored in the nodes of the trees, and words are output with uniform probability. The product of the probabilities with which each step is taken through the trees, over all such steps, equals $1/|L_n|$. How much can that probability change by truncating the values of the weights? If we have k significant bits at each node, then the probability can change at most by a factor of $1 + O(2^{-k})$ at each step, and the algorithm is linear time, so that $O(n)$ such steps are taken. Hence the maximum perturbation in the probability that any particular word is generated is at most by a factor of $\lambda_k = (1 + O(2^{-k}))^{O(n)} = 1 + O(n2^{-k})$. Taking $k = (1 + \alpha) \log_2 n$ for any constant $\alpha \geqslant 0$ makes $\lambda_k = 1 + O(n^{-\alpha})$, which converges to 1 as n gets large. Since we are truncating from at most n bits to k bits, our new floating point representation has a k bit mantissa and at most at $\log_2 n$ bit exponent, so that the storage requirement can be reduced to $O(n^2 \log n)$ bits; each word of length n is then generated with probability $|L_n|^{-1} (1 + O(n^{-\alpha}))$ in $O(n)$ time.

Acknowledgement

I would like to thank Ira Gessel, Dan Greene and Tim Hickey for various helpful comments and suggestions.

References

[1] D.B. Arnold and M.R. Sleep, Uniform random generation of balanced parenthesis strings, *ACM Trans. Programming Language Systems* **2** (1980) 122–128.

[2] R.L. Graham, D.E. Knuth and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science* (Addison-Wesley, New York, 1989).

[3] D.H. Greene, Labelled formal languages and their uses, Ph. D. Thesis Rept. STAN-CS-83-982, Stanford University, 1983.

[4] T. Hickey and J. Cohen, Uniform random generation of strings in a context-free language, SIAM J. Comput. **12** (4) (1983) 645–655.