Theoretical Computer Science 43 (1986) 169–188
North-Holland

RANDOM GENERATION OF COMBINATORIAL STRUCTURES FROM A UNIFORM DISTRIBUTION

Mark R. JERRUM
Department of Computer Science, University of Edinburgh, Edinburgh EH9 3JZ, United Kingdom

Leslie G. VALIANT *
Aiken Computation Laboratory, Harvard University, Cambridge, MA 02138, U.S.A.

Vijay V. VAZIRANI **
Computer Science Department, Cornell University, Ithaca, NY 14853, U.S.A.

Communicated by M.S. Paterson
Received July 1985
Revised November 1985

Abstract. The class of problems involving the random generation of combinatorial structures from a uniform distribution is considered. Uniform generation problems are, in computational difficulty, intermediate between classical existence and counting problems. It is shown that *exactly* uniform generation of 'efficiently verifiable' combinatorial structures is reducible to approximate counting (and hence, is within the third level of the polynomial hierarchy). Natural combinatorial problems are presented which exhibit complexity gaps between their existence and generation, and between their generation and counting versions. It is further shown that for self-reducible problems, *almost* uniform generation and *randomized* approximate counting are inter-reducible, and hence, of similar complexity.

CR Categories. F.1.1, F.1.3, G.2.1, G.3

1. Introduction

A large class of computational problems can be viewed as the seeking of partial information about a relation which associates *problem instances* with a set of *feasible solutions*. Suppose Σ is a finite alphabet in which we agree to encode both problem instances and solutions. A relation $R \subseteq \Sigma^* \times \Sigma^*$ can be interpreted as assigning, to each problem instance $x \in \Sigma^*$, a set of solutions $\{y \in \Sigma^* : xRy\}$. (For technical reasons, we shall assume that this solution set is always finite.) As a paradigm, we might take the relation which associates, with each undirected graph G, the set of

* Supported in part by National Science Foundation Grant MCS-83-02385.
** Supported by NSF Grant 85-03611, and an IBM Faculty Development Award.

1-factors (perfect matchings) of G:

$$R = \{\langle x, y\rangle : x \in \Sigma^* \text{ is an encoding of a graph } G,\ y \in \Sigma^* \text{ is an encoding of a 1-factor of } G\}.$$

(A 1-*factor* of a graph G is a spanning subgraph of G in which every vertex has degree 1.) To each relation of the above form, there correspond a number of naturally defined problems, the classical ones being existence, construction, and counting. This paper introduces a fourth kind, namely uniform generation, and investigates its relationship with the classical problem classes. A formal definition of the four problem classes is given below, each definition being followed, in parentheses, by the interpretation of the problem in the paradigmatic case of the 1-factor relation. Throughout, $x \in \Sigma^*$ represents a problem instance.

(1) *Existence*: Does there exist a word $y \in \Sigma^*$ usch that xRy? (Does the graph G contain a 1-factor?)

(2) *Construction*: Exhibit a word $y \in \Sigma^*$ satisfying xRy, if such exists. (Construct a 1-factor of G.)

(3) (*Uniform*) *Generation*: Generate uniformly, at random, a word $y \in \Sigma^*$ satisfying xRy. (Generate, at random, a 1-factor of the graph G. Each 1-factor is to appear with equal probability.)

(4) *Counting*: How many words $y \in \Sigma^*$ satisfy xRy? (How many 1-factors does G possess?)

For a given relation R, we may thus speak of the existence, construction, etc. problem associated with R. The unifying view of combinatorial problems presented above has appeared previously in the literature in the form of the *string relations* of [4], and the *search functions* of [13]. It is used here to relate generation problems, which are the main subject of the paper, to more familiar combinatorial problems such as existence and counting.

Previous papers have concentrated on particular instances of the uniform generation problem [1, 2, 16]. In the present paper, however, an attempt is made to analyse the complexity of generation problems *as a class*. Because these problems inherently involve randomization, we employ, as our model of computation, the *probabilistic Turing machine* (PTM) which is able to make random transitions according to the fall of a fair coin. A generation problem is considered to be tractable if it can be solved (in a sense which is made precise in the next section) by a PTM running in polynomial time.

A relation $R \subseteq \Sigma^* \times \Sigma^*$ is a *p-relation* if it can be 'checked fast'. Formally we require that

(1) there exists a polynomial $p(n)$ such that $\langle x, y\rangle \in R \Rightarrow |y| \leq p(|x|)$;

(2) the predicate $\langle x, y\rangle \in R$ can be tested in deterministic polynomial time.

Let $R \subseteq \Sigma^* \times \Sigma^*$ be a p-relation. It is not difficult to show that the generation problem associated with R can be solved by a polynomial time bounded PTM equipped with a $\#$P-oracle. (Essentially, $\#$P is the class of counting problems associated with p-relations [14].) We might summarize this fact informally by saying

that uniform generation is no more difficult than counting. Two pieces of evidence are provided in the paper to support the claim that generation is easier, in some sense, than counting. Firstly, it is shown that the generation problem associated with a p-relation R can be solved by a polynomial time bounded PTM equipped with a Σ_2^{P}-oracle (see [11] for a description of the polynomial hierarchy). The class $\#\mathrm{P}$, on the other hand, is not known to be contained within *any* level of the polynomial hierarchy. This containment result is akin to, and indeed rests upon, a result of Stockmeyer regarding approximate counting [12]. Secondly, a relation is presented for which the associated generation problem is solvable in polynomial time, whereas the associated counting problem is $\#\mathrm{P}$-complete. The relation in question is the one which associates each DNF Boolean formula with its set of satisfying assignments. Thus, while counting the satisfying assignments to a DNF formula is apparently computationally intractable, the task of generating random satisfying assignments is feasible. This result parallels one in [7], which exhibits a 'fully-polynomial randomized approximation scheme' for estimating the number of satisfying assignments to a DNF formula.

Evidence is also presented of a complexity gap between existence (or indeed construction) and uniform generation. (Clearly, generation is at least as difficult as construction.) We show that the existence of a polynomial time bounded PTM for uniformly generating cycles in a directed graph would imply that $\mathrm{NP} = \mathrm{RP}$. (RP is the class of decision problems which can be solved in polynomial time by a probabilistic algorithm with one-sided error probability; it is the same class that, in [5], is referred to as VPP.) Thus, it is rather unlikely that uniform generation of cycles in a directed graph can be accomplished in polynomial time, whereas the detection of cycles is an easy matter. Thus we may say, informally, that generation is strictly harder than existence.

Theorem 3.3 of the paper essentially reduces uniform generation to approximate counting. The converse reduction is clearly not possible (at least under a strict definition of approximate counting in which accuracy and termination are guaranteed) since a random process cannot achieve a deterministic requirement with certainty. The final section of the paper considers the consequences of relaxing the notion of approximate counting to require correctness within prescribed bounds only most of the time. It is shown that this notion of randomized approximate counting is inter-reducible with almost uniform generation. The equivalence holds for the class of self-reducible relations in [8] that contains many natural problems. Results related to the ones in this section have also been obtained by Broder.

2. Probabilistic Turing machines

The model of computation we employ is the probabilistic Turing machine, which was introduced in [5]. A probabilistic Turing machine (PTM) is a Turing machine [6, p. 147] equipped with an output tape and having distinguished coin-tossing

states. For each coin-tossing state, and each tape symbol scanned by the tape head, two possible transitions of the machine are specified. The computation of a PTM is deterministic, except when the PTM is in a coin-tossing state, in which case the next transition is decided by the toss of a fair coin. If the PTM reaches an accepting state, the output of the machine is just the contents of the output tape.

Gill [5] viewed PTM's as language recognizers; in this paper, however, PTM's will be used to generate random outputs, with probability distribution depending on the input x, and on some underlying relation R. We say that the PTM M is a (*uniform*) *generator* for the relation $R \subseteq \Sigma^* \times \Sigma^*$ iff

(1) there exists a function $\varphi \in \Sigma^* \to (0, 1]$ such that, for all $x, y \in \Sigma^*$,

$$\Pr(\text{given input } x,\ M \text{ outputs } y) = \begin{cases} 0 & \text{if } \langle x, y\rangle \notin R, \\ \varphi(x) & \text{if } \langle x, y\rangle \in R; \end{cases}$$

(2) for all inputs $x \in \Sigma^*$ such that $\{y \in \Sigma^* : xRy\}$ is nonempty,

$$\Pr(M \text{ accepts } x) \geqslant \tfrac{1}{2}.$$

Informally, M generates only words in the solution set of x, and each word in the solution set has an equal probability of being selected. Moreover, the probability that M will produce some output is bounded away from zero. (It is easily checked that the constant $\frac{1}{2}$ in the definition is arbitrary and may be replaced by any number strictly greater than 0 and strictly less than 1.) Note that a machine M, run on input x, signals the fact that the solution set $\{y \in \Sigma^* : xRy\}$ is empty by never accepting.

It is easy to see that using j tosses of an unbiased coin such a machine can simulate any biased coin for which the probability of landing 'heads' is of the form $i2^{-j}$ $(0 \leqslant i \leqslant 2^j)$ and that no other types of coins can be simulated. This restriction on realisable branching probabilities can sometimes be inconvenient. Indeed, the construction of PTM's to compute specified relations would be made easier, and some proofs involving PTM's made simpler, if the definition of PTM were extended to allow more general coin-tossing states. One possibility is to use a biased coin to determine the next transition of the machine—the bias of the coin being a ratio of two integers computed previously by the machine. The only objection to this extension is that it violates the idea of each computational step of a Turing machine being bounded, that is, being implementable by some fixed hardware in constant time. Once the decision has been made to use an unbiased coin, the possibility of M sometimes not accepting its input has to be allowed. For if M were to accept x for all possible sequences of coin tosses, then each word $y \in \Sigma^*$ would be output by M with a probability of the form $i2^{-j}$ for some natural numbers i and j. Thus, for example, M could not compute a relation R for which $|\{y \in \Sigma^* : xRy\}| = 3$ for some input x. Clearly, the model of computation would then be too restrictive.

We say that a PTM M is $f(n)$ time-bounded iff, for all natural numbers n and for all inputs $x \in \Sigma^n$, every accepting computation of M halts within $f(n)$ steps. A PTM M is *polynomially time-bounded* if there exists a polynomial $p(n)$ such that M is $p(n)$ time-bounded. At first sight, this definition may appear unnecessarily

severe—we might be tempted to relax it to 'the average number of steps in an accepting computation is bounded by $f(n)$'. The objection to this relaxation is that, although the average length of an accepting computation is short, there may be some words that can only be output by M after a very long computation. That is, solutions might exist which cannot be generated without a prohibitively long delay. Again, technical complications are introduced into proofs by the severity of the definition. In return, the results obtained are seen to relate to an unimpeachable model of resource-bounded computation.

3. Uniform generation and the polynomial hierarchy

For the main result of this section, we make use of a theorem of Stockmeyer. If α, β, r are positive real numbers, with $r \geqslant 1$, we say that β approximates α within ratio r if $\beta r^{-1} \leqslant \alpha \leqslant \beta r$.

Theorem 3.1. *Let $f \in \Sigma^* \to \mathbb{N}$ be a member of $\#\mathrm{P}$. Then there exists a deterministic TM M, equipped with a Σ_2^{p}-oracle, which for all inputs $\langle x, \varepsilon \rangle \in \Sigma^* \times \mathbb{R}^+$ produces an output $M(x, \varepsilon)$ approximating $f(x)$ within ratio $1+\varepsilon$. Moreover, the run-time of M is bounded by a polynomial in $|x|$ and $1/\varepsilon$. (A definition of Σ_2^{p} is given in* [4, p. 162].)

Proof. See [12]. The proof relies on Sipser's [10] technique for estimating the size of a set by means of universal hash functions. □

We also require an easy technical lemma which ensures that adequate approximations to numbers in a certain interval can be found within a small set of rationals.

Lemma 3.2. *Let m be a positive integer. There exists a set of rational numbers A, of cardinality $4m^2$, satisfying*

(1) *each member of A is of the form $i2^{-2m}$ for some integer i in the range $[0, 2^{2m}]$;*

(2) *for each real number α, $2^{-m} \leqslant \alpha < 1$, there is a rational $q \in A$ such that $q \leqslant \alpha$, and q approximates α within ratio $1+1/m$.*

Proof. Let $k = 4m^2$, $r = 2^{1/2m}$, and $q_i = \lceil r^i \rceil 2^{-2m}$ for $1 \leqslant i \leqslant k$. We claim that the set $A = \{q_i : 1 \leqslant i \leqslant k\}$ satisfies the two properties required by the statement of the lemma. That A satisfies property (1) is immediate. To verify property (2), suppose α satisfies $2^{-m} \leqslant \alpha < 1$, and choose i such that $q_i \leqslant \alpha < q_{i+1}$. Note that $i \geqslant 2m^2$ since, for $i < 2m^2$, $q_{i+1} \leqslant 2^{-m} \leqslant \alpha$. We need to show that q_i approximates α within ratio $1+1/m$. Consider the following chain of inequalities:

$$\alpha / q_i < \frac{q_{i+1}}{q_i} = \frac{\lceil r^{i+1} \rceil}{\lceil r^i \rceil} < \frac{(r^{i+1}+1)}{r^i} = r + r^{-i}. \tag{1}$$

By applying the binomial theorem to $(1+1/2m)^{2m}$, we obtain

$$r<1+\frac{1}{2m}.$$

Also, since $i \geq 2m^2$,

$$r^{-i} \leq 2^{-m} \leq \frac{1}{2m}.$$

Substituting these upper bounds in (1) completes the verification. □

Theorem 3.3. *Let $R \subseteq \Sigma^* \times \Sigma^*$ be a p-relation. Then there exist uniform generators for R of the following types:*

(1) *a polynomial time bounded PTM equipped with a $\#$P-oracle,*

(2) *a polynomial time bounded PTM equipped with a Σ_2^{P}-oracle.*

Proof. Since R is a p-relation, there is a polynomial $p(n)$ such that, for all $x, y \in \Sigma^*$, $xRy \Rightarrow |y| \leq p(|x|)$. By padding words which are deficient in length, we can, without loss of generality, assume the stronger condition: $xRy \Rightarrow |y| = p(|x|)$. As a further simplification, we shall assume that problem instances and solutions are encoded in binary, i.e., that $\Sigma = \{0, 1\}$.

The proof is by reduction of uniform generation to approximate counting. The reduction is conceptually simple and can be informally described in a few lines. Consider the PTM M which operates in the following manner. On input $x \in \Sigma^n$, M generates a sequence $y_1, \ldots, y_{p(n)}$ of $p(n)$ binary digits, in which each digit is determined by tossing a fair coin. The distribution of the resulting word $y = y_1 \ldots y_{p(n)}$ is obviously uniform over the set $\Sigma^{p(n)}$. The machine M then tests, in polynomial time, whether xRy; if the test succeeds, M outputs y and accepts. Clearly, M generates each word in the set $\{y \in \Sigma^* : xRy\}$ with equal probability. Unfortunately, since accepting computations of M may form only a small proportion of the total, the probability that M produces no output may be very close to 1.

Suppose, however, that each time M enters a coin-tossing state, we have available some information concerning the number of accepting configurations which can be reached given the fall of the coin. Then we can improve the chances of arriving at an accepting configuration by throwing an unfair coin which favours the outcome which leads to the larger number of accepting configurations. In fact, given exact information about the number of accepting configurations of M which can be reached from a given configuration, we can so choose the biasing of the coin that

- only accepting configurations can be reached;
- each accepting configuration is reached with equal probability.

Part (1) of the theorem is proved by observing that the required information can be obtained using a $\#$P-oracle.

If the information about the number of reachable accepting configurations of M is approximate, but sufficiently accurate, we can use a biased coin to arrive at

accepting configurations of M with roughly uniform probability. Since the bias of the coin is prescribed, the probability that M will reach a certain accepting configuration is known a posteriori. By retaining the output of M with probability inversely proportional to the a posteriori probability and discarding it otherwise, we can force a uniform distribution on the accepting configurations. (A similar technique is used in [1] to generate integers with known factorization.) Part (2) of the theorem is completed by appealing to Theorem 3.1.

The detailed proof of the theorem involves a number of technicalities which arise from the inability of a PTM to branch with other than even probabilities. Suppose $R \subseteq \Sigma^* \times \Sigma^*$ is a p-relation, and $x \in \Sigma^n$. Let $m = p(n)$, $r = (1+1/m)$, and A be a set of rationals satisfying the conditions of Lemma 3.2. Define the extension counting function $\mathrm{Ext}_R \in (\Sigma^* \times \Sigma^* \to \mathbb{N})$ by

$$\mathrm{Ext}_R(x, w) = |\{z \in \Sigma^* : \langle x, wz \rangle \in R\}|.$$

It will be shown that the procedure UGEN (Fig. 1), when called with appropriately chosen parameters, is a uniform generator for the relation R. The procedure UGEN invokes a function APPROXCOUNT which is assumed to meet the following specification: For all $x, w \in \Sigma^*$, $\varepsilon \in \mathbb{R}^+$, APPROXCOUNT$(x, w, \varepsilon)$ approximates $\mathrm{Ext}_R(x, w)$ within ratio $1+\varepsilon$.

```
          Procedure UGEN(x, w, φ);
          begin
(1)         N_0 := APPROXCOUNT(x, w0, 1/m); N_1 := APPROXCOUNT(x, w1, 1/m);
(2)         if |w| = m then with probability φ: output w
(3)         else if N_0 = 0 then UGEN(x, w1, φ)
(4)         else if N_1 = 0 then UGEN(x, w0, φ)
            else begin
(5)           α_0 := N_0/(N_0 + N_1); α_1 := N_1/(N_0 + N_1);
(6)           choose q_0 ∈ A with α_0 r^{-1} < q_0 ≤ α_0;
(7)           choose q_1 ∈ A with α_1 r^{-1} < q_1 ≤ α_1;
(8)           either with probability q_0: UGEN(x, w0, φ/q_0)
              or with probability q_1: UGEN(x, w1, φ/q_1)
            end
          end
```

Fig. 1. The uniform generation procedure, UGEN.

For many choices of the parameters x, w, and φ, the procedure UGEN will fail during execution. Failure will occur if the branching probability φ in line 2 is not in the range [0, 1] or is not expressible as a fraction whose denominator is a power of two. (These are the conditions which must be met if the randomized branch is to be realized using an experiment with a fair coin.) Suppose, for the time being, that for given $x, w \in \Sigma^*$ we can choose $\varphi \in \mathbb{R}^+$ so that the procedure call UGEN(x, w, φ) is guaranteed to execute without error; let the random variable $Y = \Sigma^* \cup \{\perp\}$ be the output resulting from the procedure call (we use the symbol

$\perp$ to stand for 'no output'). Under the assumption that UGEN(x, w, φ) executes without error, an easy induction on $|y|-|w|$ establishes that, for all $y \in \Sigma^*$,

$$\Pr(Y=y)=\begin{cases}\varphi & \text{if } xRy \text{ and } w \text{ is an initial segment of } y,\\ 0 & \text{otherwise.}\end{cases}$$

Thus, in particular, setting $w=\Lambda$ (the empty word) and $\varphi=\varphi_0$:

$$\Pr(Y=y)=\begin{cases}\varphi_0 & \text{if } xRy,\\ 0 & \text{otherwise.}\end{cases}$$

The crucial aspect of the proof is showing that the parameter φ_0 can be chosen so that no failure occurs in line 2, and moreover that $\Pr(Y=\perp)$ is bounded away from 1.

To ensure that no failure occurs in line 2, it is enough to take $\varphi_0=2^t\prod_{q\in A} q^m$, where $t\in\mathbb{N}$ is sufficiently small so that

$$\varphi_0 \operatorname{Ext}_R(x,\Lambda)\leqslant e^{-3}. \tag{2}$$

We claim that, at every level of recursion, the program variables φ and w satisfy

$$\varphi \operatorname{Ext}_R(x,w)\leqslant r^{3(|w|-m)}, \tag{3}$$

and hence, at the greatest depth of recursion, $\varphi\leqslant 1$ as desired. (The other fact we require, namely that, at all levels of recursion, φ is a rational whose denominator is a power of 2, is clear from the construction of φ_0.) Equation (3) can be verified by induction on $|w|$. To establish the base case, $w=\Lambda$, note that:

$$\varphi \operatorname{Ext}_R(x,w)=\varphi_0 \operatorname{Ext}_R(x,\Lambda)\leqslant e^{-3}\leqslant\left(1+\frac{1}{m}\right)^{-3m}=r^{3(|w|-m)}.$$

For the induction step, we observe, from the specification of APPROXCOUNT, that N_0 and N_1 respectively approximate $\operatorname{Ext}_R(x,w0)$ and $\operatorname{Ext}_R(x,w1)$ within ratio r. Hence, by construction, α_0 approximates $\operatorname{Ext}_R(x,w0)/\operatorname{Ext}_R(x,w)$ within ratio r^2 and q_0 approximates the same quantity within ratio r^3. Now let φ' and w' denote the values of the corresponding program variables on the *succeeding* level of recursion and assume, without loss of generality, that $w'=w0$. Then

$$\begin{aligned}\varphi' \operatorname{Ext}_R(x,w') &= (\varphi/q_0)\operatorname{Ext}_R(x,w0)\\ &\leqslant \varphi r^3 \operatorname{Ext}_R(x,w)\\ &\leqslant r^3 r^{3(|w|-m)} \quad \text{(induction hypothesis)}\\ &= r^{3(|w'|-m)}.\end{aligned}$$

This establishes the induction step, and hence, the inequality (3).

Finally, note that $\Pr(Y=\perp)\leqslant 1-\varphi_0 \operatorname{Ext}_R(x,\Lambda)$. When choosing φ_0 we are constrained only by inequality (2). By initially estimating $\operatorname{Ext}_R(x,\Lambda)$ using a call to APPROXCOUNT, we can choose φ_0 so that the two sides in (2) differ by at most a factor of $2r$. For this choice of φ_0, $\Pr(Y=\perp)\leqslant 1-1/2re^3$.

Clearly, the calls to the procedure APPROXCOUNT in line 2 can be dealt with by a #P-oracle (which would in fact yield exact values for $\operatorname{Ext}_R(x,w0)$ and

$\mathrm{Ext}_R(x, w1)$). This dispenses with part (1) of the theorem. More interestingly, since the only requirement of ApproxCount is that it approximate $\mathrm{Ext}_R(x, w)$ within ratio $1+1/m$, and since the latter, as a function of x and w, is in $\#\mathrm{P}$, we know from Theorem 3.1 that a Σ_2^{P}-oracle also suffices. This observation immediately gives part (2) of the theorem. □

In Section 6 a notion of *almost uniform generation* is introduced which allows the probability distribution on outputs to deviate slightly from the ideal; the extent of the deviation is controlled by an input parameter called the *tolerance*. It is possible to locate this relaxed version of generation lower in the polynomial hierarchy than the strict version already considered. The following variant of Theorem 3.1 is implicit in [10, 12]; alternatively, it can be easily derived using the probabilistic bisection technique of [15].

Theorem 3.4. *Let $f \in \Sigma^* \to \mathbb{N}$ be a member of $\#\mathrm{P}$. Then there exists a PTM M, equipped with an* NP- (*i.e.*, Σ_1^{P}-)*oracle, which for all inputs $\langle x, \varepsilon\rangle \in \Sigma^* \times \mathbb{R}^+$ produces an output $M(x, \varepsilon)$ (a random variable of the coin-tossing sequence of M) satisfying*

$$\Pr(M(x, \varepsilon) \text{ approximates } f(x) \text{ within ratio } (1+\varepsilon)) \geq \tfrac{3}{4}.$$

Moreover, the run-time of M is bounded by a polynomial in $|x|$ and $1/\varepsilon$.

Theorem 3.5. *Let $R \subseteq \Sigma^* \times \Sigma^*$ be a p-relation. There exists a polynomial time bounded PTM M, equipped with an* NP*-oracle, which is an almost uniform generator for R (in the sense of Section* 6).

Sketch of proof. A slight modification to the proof of Theorem 3.3 yields a reduction from almost uniform generation to randomized approximate counting. The result then follows from Theorem 3.4. □

4. An instance where uniform generation is easier than counting

The previous section provided circumstantial evidence that uniform generation problems as a class may be 'easier' than counting problems. In this section, we consider a particular relation for which the associated counting problem is apparently intractable, whereas the associated generation problem is efficiently soluble. The relation in question associates each Boolean formula F in disjunctive normal form (DNF) with the set of satisfying assignments to F. It is easy to demonstrate that the problem of counting the satisfying assignments to a DNF-formula is $\#$P-complete, and hence, unlikely to be soluble in deterministic polynomial time. (A description of $\#\mathrm{P}$ and its completeness class can be found in [4, 14].) In contrast, a simple and efficient algorithm for generating random assignments to a DNF-formula exists and is presented in this section.

The $\#\mathrm{P}$-completeness result is obtained by exhibiting a reduction from $\#$Sat, the problem of counting the number of satisfying assignments to a Boolean formula

in conjunctive normal form (CNF). Simon [9] showed #SAT to be #P-complete by a slight modification to the generic transformation used to establish Cook's Theorem. Suppose now that F is an instance of #SAT. The formula F can be transformed, using De Morgan's laws, into an equivalent length DNF-formula for $\neg F$, the complement of F. This having been done, it merely needs to be noted that the number of satisfying assignments to F plus the number of satisfying assignments to $\neg F$ is equal to 2^k, where k is the number of variables occurring in F.

The proposed method for randomly generating satisfying assignments to a DNF-formula closely follows the Monte-Carlo algorithm, presented in [7], for estimating their number. (Indeed, it will become apparent later that an *almost* uniform generator for satisfying assignments directly follows from the Karp-Luby algorithm [7] via the reduction presented in the previous section. However, the method described in this section has the advantage of yielding an *exactly* uniform generator for the problem.) Suppose that $F = F_1 \vee F_2 \vee \cdots \vee F_m$ is a DNF-formula in the set of variables X, with each F_i being a conjunction of a number of literals. Let $S_j \subseteq \{0,1\}^X$, $1 \leq j \leq m$, be the set of satisfying assignments to the disjunct F_j and $S = \bigcup_k S_k$ the set of satisfying assignments to F itself. The task is to select, uniformly at random, a member of the set S. The algorithm for accomplishing the task is sketched in Fig. 2.

```
(1)   for i := 1 to m do begin
(2)     select an integer j ∈ [1, m] randomly, but nonuniformly, such that Pr(j = c)
          = |S_c| / Σ_k |S_k|;
(3)     select a ∈ S_j, uniformly at random;
(4)     N := |{k ∈ [1, m]: a ∈ S_k}|;
(5)     with probability 1/N: output a and halt
      end
```

Fig. 2. Algorithm for generating a satisfying assignment to a DNF-formula.

Consider one iteration of the **for** loop. Let j_0 be any integer in the range $[1, m]$, and a_0 any element of S_{j_0}. After line 3 of the **for** loop is executed, the probability that the variables j and a have the values j_0 and a_0 respectively is $(\sum_k |S_k|)^{-1}$, independent of the choice of j_0 and a_0. Thus, the probability that the variable a takes the value a_0 is $N/\sum_k |S_k|$, where $N = |\{k \in [1, m]: a_0 \in S_k\}|$, and the probability of a_0 being output in line 5 is $(\sum_k |S_k|)^{-1}$, independent of the choice of a_0. The algorithm therefore generates each satisfying assignment to F with equal probability. It remains to check that the probability that the algorithm generates no output is bounded away from one. The probability that, on a particular iteration, some assignment is output, is greater than $1/m$. The probability that m iterations occur with no output taking place is therefore less than $(1-1/m)^m$, which is bounded above by e^{-1}.

The algorithm given in Fig. 2 is not, as it stands, directly implementable on a PTM (the branching probabilities are not of the required form). This objection can be dealt with by slightly modifying the selection probabilities used in line 2 of the

algorithm. Let $t \in \mathbb{N}$ be the smallest integer satisfying $2^t \geq \sum_k |S_k|$, and let the integer j in line 2 of the algorithm be chosen with probabilities given by

$$\Pr(j = c) = 2^{-t}|S_c|, \quad \text{for all } c,\ 1 \leq c \leq m.$$

There is now a nonzero probability (less than $\frac{1}{2}$) that j will be undefined, in which case the algorithm skips immediately to the next iteration of the **for** loop. The probability that in one iteration of the **for** loop no output takes place is now bounded above by $1 - 1/2m$ and the probability that the algorithm terminates with no output taking place is bounded above by $e^{-1/2}$.

5. Evidence that uniform generation is harder than construction

Generation problems differ from construction problems in requiring uniformity of the probability distribution on the space of possible outputs. It is natural to ask whether this additional requirement makes uniform generation strictly harder than construction. The following theorem suggests that there are naturally occurring relations for which the associated construction and generation problems are of widely differing complexities. Let GENCYCLE be the following problem:

Input: Directed graph G.

Output: A cycle selected uniformly, at random, from the set of all directed (simple) cycles of G.

Clearly, the problem of constructing an arbitrary cycle in a directed graph is easily solved in polynomial time.

Theorem 5.1. *Suppose there exists a polynomial time bounded PTM which solves the problem* GENCYCLE. *Then* NP = RP.

Proof. It is sufficient to deduce, assuming the condition of the theorem, that RP contains some NP-complete problem. We choose to work with the problem DHC of determining whether a directed graph $G = \langle V, E \rangle$ contains a Hamiltonian cycle [4]. Let $G' = \langle V', E' \rangle$ be the directed graph derived from G by replacing each edge of G by the 'chain of diamonds' illustrated in Fig. 3. The length of the chain is chosen to be $k = \lceil n \log n \rceil$, where n is the number of vertices in G.

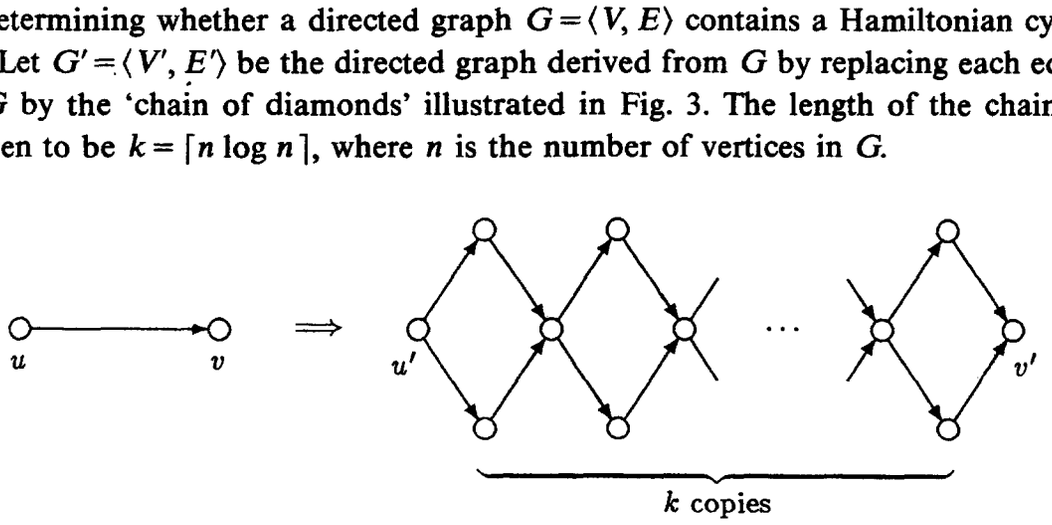

Fig. 3. The transformation applied to each edge of G.

Formally, the vertex and edge sets of the transformed graph G' are given by

$$V' = V \cup E \times \{0, \ldots, 3k-2\},$$

$$E' = \{\langle u, \langle e, 0\rangle\rangle, \langle u, \langle e, 1\rangle\rangle, \langle\langle e, 3k-3\rangle, v\rangle, \langle\langle e, 3k-2\rangle, v\rangle : e = \langle u, v\rangle \in E\}$$

$$\cup \{\langle\langle e, 3i\rangle, \langle e, 3i+2\rangle\rangle, \langle\langle e, 3i+1\rangle, \langle e, 3i+2\rangle\rangle, \langle\langle e, 3i+2\rangle, \langle e, 3i+3\rangle\rangle,$$

$$\langle\langle e, 3i+2\rangle, \langle e, 3i+4\rangle\rangle : e \in E, 0 \leqslant i \leqslant k-2\}.$$

Clearly, the transformed graph G' contains a cycle of length $2kn$ if and only if the original graph G contains a Hamiltonian cycle. Now, if G' contains a cycle of length $2kn$, then it contains at least 2^{kn} cycles of this length. Moreover, it is easy to check that the total number of cycles in G' which have length shorter than $2kn$ is bounded by $n^n 2^{k(n-1)}$, which in turn, by choice of k, is bounded by 2^{kn}. Thus, if G is Hamiltonian, the probability that a randomly generated cycle of G' has length $2kn$ is at least $\frac{1}{2}$, whereas, if G is not Hamiltonian, the probability is 0. Membership of DHC in RP is immediate from this observation. □

It would be interesting to find other examples of natural problems which exhibit a complexity gap between their construction and generation variants. One possible candidate is the problem of uniformly generating a 1-factor in an undirected graph. This problem is not known to be polynomial time solvable, but neither is there any convincing evidence of intractability.

6. The relationship between almost uniform generation and randomized approximate counting

Sections 3 and 4 suggest that uniform generation of combinatorial structures is in some way related to approximate counting of structures as studied in [7, 12]. For problems which are self-reducible in the sense of Schnorr, this connection can be formalized. In order to achieve this, however, it appears necessary to slightly weaken our strict notion of uniform generation to one of *almost uniform* generation. For all practical purposes an almost uniform generator is just as good as an exactly uniform generator—viewed as black boxes, the two would be impossible to tell apart by an experiment of polynomially bounded duration. The main result of this section is that approximate counting, in the sense of Karp and Luby, is of equivalent complexity to almost uniform generation. As a corollary, we demonstrate how it is possible to transform a generator whose output distribution is rather far from uniform, to one which is much closer to the ideal uniform distribution. The transformation involves little degradation in efficiency of the generator. In a similar way, a fairly inaccurate procedure for approximately counting combinatorial structures can be transformed into one which provides more accurate results.

Following [8] we say that a relation $R \subseteq \Sigma^* \times \Sigma^*$ is *self-reducible* iff

(1) there exists a polynomial time computable function $g \in \Sigma^* \to \mathbb{N}$ such that $xRy \Rightarrow |y| = g(x)$;

(2) there exist polynomial time computable functions $\psi \in \Sigma^* \times \Sigma^* \to \Sigma^*$ and $\sigma \in \Sigma^* \to \mathbb{N}$ satisfying

$$\sigma(x) = O(\log |x|),$$
$$g(x) > 0 \Rightarrow \sigma(x) > 0 \quad \forall x \in \Sigma^*,$$
$$|\psi(x, w)| \leqslant |x| \quad \forall x, w \in \Sigma^*,$$

and such that, for all $x \in \Sigma^*$, $y = y_1 \ldots y_n \in \Sigma^*$,

$$\langle x, y_1 \ldots y_n \rangle \in R \Leftrightarrow \langle \psi(x, y_1 \ldots, y_{\sigma(x)}), y_{\sigma(x)+1} \ldots y_n \rangle \in R.$$

Intuitively, self-reducibility captures the idea that the solution set associated with a given instance of a problem can be expressed in terms of the solution sets of a number of smaller instances of the same problem. The function g gives the length of the solutions to instances and σ gives the granularity of solutions in the following sense. Given an instance x and initial segment w (of length $\sigma(x)$) of any solution to x, ψ gives an instance x' whose solutions are exactly those words which, when concatenated with w, form solutions to x. Very many naturally occurring relations are self-reducible; examples include 1-factors in an undirected graph and satisfying assignments to a CNF- or DNF-formula. In fact, problems which cannot be formulated in a self-reducible way seem to be the exception rather than the rule.

In order to formalize the idea of almost uniform generation, we consider PTM's which take, in addition to the usual input $x \in \Sigma^*$, a positive real *tolerance* ε, $0 \leqslant \varepsilon < 1$ (on the input tape, the parameter ε might be denoted as the reciprocal of a specified integer). A PTM M is an almost uniform generator for the relation $R \subseteq \Sigma^* \times \Sigma^*$ iff

(1) there exists a function $\varphi \in \Sigma^* \to (0, 1]$ such that, for all inputs $\langle x, \varepsilon \rangle \in \Sigma^* \times \mathbb{R}^+$ to M and for all words $y \in \Sigma^*$,

$$\langle x, y \rangle \notin R \Rightarrow \Pr(M \text{ outputs } y) = 0,$$
$$\langle x, y \rangle \in R \Rightarrow (1+\varepsilon)^{-1}\varphi(x) \leqslant \Pr(M \text{ outputs } y) \leqslant (1+\varepsilon)\varphi(x);$$

(2) for all inputs $\langle x, \varepsilon \rangle$ such that $\{y \in \Sigma^* : xRy\}$ is nonempty, $\Pr(M \text{ accepts } \langle x, \varepsilon \rangle) \geqslant \frac{1}{2}$.

A possible alternative to the second part of the definition is to insist that M always accepts its input. The definition chosen has the advantage of including exactly uniform generation as a special case (the alternative definition presumably does not have this property). We shall say that an almost uniform generator is *fully-polynomial* (f.p.) if its execution time is bounded by a polynomial in $|x|$ and $\log \varepsilon^{-1}$ (the inclusion of the logarithm means that an f.p. almost uniform generator can, at modest computational expense, achieve an output distribution which is very close to uniform).

The notion of randomized approximate counting we employ is the same as that in [7]. Suppose $f \in \Sigma^* \to \mathbb{N}$. A *randomized approximation scheme* for f is a PTM M which for all inputs $\langle x, \varepsilon \rangle \in \Sigma^* \times \mathbb{R}^+$ produces an output $M(x, \varepsilon)$ (a random variable of the coin-tossing sequence of M) such that

$$\Pr(M(x, \varepsilon) \text{ approximates } f(x) \text{ within ratio } (1+\varepsilon)) \geqslant \tfrac{3}{4}$$

(the constant $\frac{3}{4}$ in the definition can be replaced by any number lying strictly between $\frac{1}{2}$ and 1). A randomized approximation scheme is fully-polynomial if its execution time is bounded by a polynomial in $1/\varepsilon$ and the length of x. Counting satisfying assignments to a DNF Boolean formula is an example of a combinatorial enumeration problem which is apparently hard to solve exactly, but for which an f.p. randomized approximation scheme exists [7]. Note the important difference in the definitions of f.p.: for counting we allow inverse polynomial errors, while for generation we tolerate only inverse exponential errors.

Our eventual aim is to show that, for self-reducible problems, the time-complexities of almost uniform generation and randomized approximate counting are within a polynomial factor of each other. We require two preparatory lemmata, the first of which is important in its own right.

Lemma 6.1 (Powering lemma for randomized approximation schemes). *Let $f \in \Sigma^* \to \mathbb{N}$, and suppose that there is an f.p. randomized approximation scheme for f. Then there exists a PTM M which on input $\langle x, \varepsilon, \delta\rangle \in \Sigma^* \times \mathbb{R}^+ \times \mathbb{R}^+$ produces an output $M(x, \varepsilon, \delta)$ (a random variable) such that*

$$\Pr(M(x, \varepsilon, \delta) \text{ approximates } f(x) \text{ within ratio } 1+\varepsilon) \geq 1-\delta.$$

Moreover, the execution time of M is bounded by a polynomial in $|x|$, $1/\varepsilon$, and $\log \delta^{-1}$.

Proof. We may assume $\delta < 1$, for the lemma is otherwise vacuously true. The operation of the machine M is as follows. On input $\langle x, \varepsilon, \delta\rangle$, M runs the postulated randomized approximation scheme $t = 12\lceil -\log \delta \rceil + 1$ times with input $\langle x, \varepsilon\rangle$; M then outputs the median of the t results. The probability that the median fails to approximate $f(x)$ within ratio $1+\varepsilon$ is bounded above by

$$\sum_{i=\frac{1}{2}(t+1)}^{t} \binom{t}{i} \left(\frac{1}{4}\right)^i \left(\frac{3}{4}\right)^{t-i}$$

By Chernoff's bound [3, p. 17], this sum is less than $e^{-t/12}$, and hence, by choice of t, less than δ. □

Suppose $R \subseteq \Sigma^* \times \Sigma^*$ is a relation. Let $N_R \in \Sigma^* \to \mathbb{N}$ be the counting function associated with R, defined by $N_R(x) = |\{y \in \Sigma^* : xRy\}|$. Note that $N_R(x) = \mathrm{Ext}_R(x, \Lambda)$.

Lemma 6.2. *Let R be self-reducible. If there exists an f.p. randomized approximation scheme for N_R, then there exists an f.p. randomized approximation scheme for* Ext_R.

Proof. Suppose that we wish to compute $\mathrm{Ext}_R(x, w)$ for $x, w \in \Sigma^*$. The idea of the proof is to use self-reducibility of R to express $\mathrm{Ext}_R(x, w)$ as $\mathrm{Ext}_R(x', w')$ for some x', w' with $|x'| \leq |x|$ and $|w'| < |w|$. By repeated application of this process, the second argument, w, can be made short and the task can be completed using a small number of calls to the randomized approximation scheme for N_R.

This algorithm is expressed more formally in Fig. 4. Let $\mathcal{A}$ be the postulated f.p. randomized approximation scheme for N_R; by Lemma 6.1, we can assume that $\mathcal{A}$ takes an extra parameter δ which controls the error probability of the scheme. The functions σ and ψ which appear in the algorithm are the ones whose existence is guaranteed by the definition of self-reducibility.

Referring to the definition of self-reducibility, it is easy to see that $\mathrm{Ext}_R(x, w)$ remains constant during execution of the **while** loop; moreover, since $|w|$ decreases at each iteration, the loop is guaranteed to terminate. Now for any $u \in S$, the probability that, on input $\langle \psi(x, wu), \varepsilon, 1/4|S| \rangle$, $\mathcal{A}$ fails to approximate $\mathrm{Ext}_R(x, wu)$ within ratio $1+\varepsilon$ is bounded by $1/4|S|$. Hence, the probability that the value returned by the algorithm fails to approximate $\mathrm{Ext}_R(x, w)$ within ratio $1+\varepsilon$ is bounded above by $\frac{1}{4}$. In other words, the algorithm of Fig. 4 is indeed a randomized approximation scheme for Ext_R. It is easy to check that the scheme is fully-polynomial. □

We are now ready to prove one half of the equivalence promised at the beginning of the section. Informally stated, a fast algorithm for approximate counting implies a fast algorithm for almost uniform generation.

Theorem 6.3. *Let R be a self-reducible p-relation. If there exists an f.p. randomized approximation scheme for N_R, then there exists an f.p. almost uniform generator for R.*

Proof. Lemma 6.2 assures us of the existence of an f.p. randomized approximation scheme, $\mathcal{A}$ say, for Ext_R. By the powering Lemma 6.1, we may assume that $\mathcal{A}$ takes as input an extra parameter δ, which controls the error probability of $\mathcal{A}$.

We construct an f.p. almost uniform generator for R which is based on the uniform generation procedure of Fig. 1. The calls to the procedure ApproxCount in line 1 are replaced by calls to $\mathcal{A}$ with inputs $\langle x, w0, 1/m, \delta \rangle$ and $\langle x, w1, 1/m, \delta \rangle$ respectively (we will see later how to choose δ appropriately). When the modified procedure Ugen is executed, the variables N_0 and N_1 respectively, may fail to approximate $\mathrm{Ext}_R(x, w0)$ and $\mathrm{Ext}_R(x, w1)$ within ratio $r = (1+1/m)$. The probability of this error occurring is dependent on the parameter δ, which, by the powering lemma, may be made exponentially small without compromising the polynomial run time of $\mathcal{A}$.

Now imagine that there is a benign spirit, which oversees the execution of the procedure Ugen. When $\mathcal{A}$ is about to return a value which is not within the proper range, the benign spirit intervenes and substitutes a value which *is* in range. With

```
while |w| > σ(x) do begin
  express w as uv with |u| = σ(x);
  w := v;
  x := ψ(x, u)
end;
S := Σ^{σ(x)−|w|};
return Σ_{u∈S} 𝒜(ψ(x, wu), ε, 1/4|S|)
```

Fig. 4. Algorithm for computing $\mathrm{Ext}_R(x, w)$.

the spirit's aid, we are again in the situation of the proof of Theorem 3.3 and UGEN becomes an *exact* uniform generator for R. For any input $x \in \Sigma^*$, each word in the set $\{y \in \Sigma^* : xRy\}$ is equally likely to appear as output of the uniform generator; let this uniform output probability be $\varphi(x)$. Note that $\varphi(x) \geq 2^{-(m+1)}$ since the generation procedure produces at most 2^m distinct outputs, and the probability of *some* output is greater than $\frac{1}{2}$.

A call to the procedure UGEN initiates $2m$ calls to $\mathscr{A}$, and an additional call to $\mathscr{A}$ is required to initialize the parameter φ_0. The probability that the spirit intervenes during a single run of the generator is therefore no greater than $(2m+1)\delta$. Hence, if the benign spirit is banished, no output probability of the generator is perturbed by more than an additive term $(2m+1)\delta$. Let $Y \in \Sigma^*$, a random variable, be the output of the generation procedure (with no intervention by the spirit) on input x. Then, for $y \in \Sigma^*$, we have

$$\langle x, y\rangle \in R \Rightarrow \varphi(x) - (2m+1)\delta \leq \Pr(Y = y) \leq \varphi(x) + (2m+1)\delta,$$

$$\langle x, y\rangle \notin R \Rightarrow \Pr(Y = y) \leq (2m+1)\delta.$$

There now is a small probability that a word y will be output which does not satisfy xRy. This fault is easily corrected by checking the condition xRy before output (the check can be performed in polynomial time since R is a p-relation). Choose ε, $0 < \varepsilon < 1$, and let y satisfy xRy. We can ensure that $\Pr(Y = y)$ approximates $\varphi(x)$ within ratio $1+\varepsilon$ by setting $\delta = \varepsilon/(2m+1)2^{m+2}$ (recall that $\varphi(x)$ is bounded below by $2^{-(m+1)}$). The run time of the approximation scheme $\mathscr{A}$ is polynomial in $\log \delta^{-1}$, and hence, the run time of the derived generator is polynomial in $\log \varepsilon^{-1}$, as required. □

As a possible application of Theorem 6.3, suppose there were an efficient randomized algorithm for estimating the number of 1-factors of a graph. Then, since the 1-factor relation is self-reducible, the problem of generating 1-factors of a graph almost uniformly would also be feasible. A second example is provided by satisfying assignments to a DNF formula. The existence of an efficient Monte-Carlo algorithm [7] for estimating the number of satisfying assignments to a DNF-formula immediately implies that the problem of generating satisfying assignments almost uniformly is solvable in polynomial time (of course, a slightly stronger result has already been obtained in Section 4).

Theorem 6.3 has a converse, which is formulated as follows.

Theorem 6.4. *Let R be a self-reducible p-relation. If there exists an f.p. almost uniform generator for R, then there exists an f.p. randomized approximation scheme for N_R.*

Proof. Let $\mathscr{G}$ be a f.p. almost uniform generator for R. A possible strategy for estimating $N_R(x)$, given $x \in \Sigma^*$, has the following outline. By making sufficiently many calls to the generator $\mathscr{G}$, an estimate can be made of the relative magnitudes of $\mathrm{Ext}_R(x, u)$ for $u \in \Sigma^{\sigma(x)}$ (the function σ here is the one whose existence is

guaranteed by the self-reducibility of R). Now choose $w \in \Sigma^{\sigma(x)}$ such that $\mathrm{Ext}_R(x, w)$ is large (choosing w in this way maximizes the accuracy of the technique). The series of experiments using $\mathcal{G}$ yields an estimate for the quotient $N_R(x)/\mathrm{Ext}_R(x, w)$. Now the denominator of the quotient can be expressed as $N_R(\psi(x, w))$, where the function ψ is as in the definition of self-reducibility; furthermore, the value of $N_R(\psi(x, w))$ can be estimated recursively. Multiplying together the two estimates yields the sought-after estimate for $N_R(x)$. We need to verify that adequate accuracy can be obtained using only a polynomially bounded number of calls to $\mathcal{G}$.

A formal description of the algorithm appears in Fig. 5. In the algorithm, g, σ, and ψ have the meanings ascribed in the definition of self-reducibility. The integer m is an upper bound on the set $\{|y| : xRy\}$; we may take $m = g(x)$. The constant c only depends on R and is chosen so that $2^{\sigma(x)} \leq |x|^c$ for all words $x \in \Sigma^*$ of sufficient length; such a constant exists by self-reducibility of R. The parameter ε controls the accuracy of the result, and is in the range $(0, 1)$.

```
        Π := 1;
        t := 180|x|^{3c} m^3 / ε^2;
        while g(x) > 0 do begin
(1)       make 3t calls to 𝒢 with input ⟨x, ε/11m⟩;
(2)       if at least t of the 3t trials yield an output
            then let S = {y_1, ..., y_t} be the first t outputs of 𝒢
            else return 0;
(3)       Let w ∈ Σ^{σ(x)} be a most commonly occurring prefix in S of length σ(x);
(4)       α := |{y ∈ S : w is an initial segment of y}|/|S|;
(5)       x := ψ(x, w);
(6)       Π := Π/α
        end;
(7)     output Π
```

Fig. 5. Algorithm for estimating $N_R(x)$.

For the rest of the proof we assume that $N_R(x) > 0$, for the algorithm is clearly correct in the case $N_R(x) = 0$. We shall say that the algorithm *runs to completion* if line 7 is reached (the other possibility is that the algorithm terminates at line 2). The value of α computed in line 4 is intended to be an approximation to $\mathrm{Ext}_R(x, w)/N_R(x)$. In fact, we shall show later that the probability that the following two events occur simultaneously is at least $\frac{3}{4}$:

'the algorithm runs to completion' (4)

and

'on every iteration of the loop, α approximates $\mathrm{Ext}_R(x, w)/N_R(x)$ within ratio $1 + \varepsilon/2m$'. (5)

Let us restrict attention to runs of the algorithm in which (4) and (5) do both hold. Let x_0 denote the initial value of the program variable x. Our aim is to show that, on termination of the algorithm, Π approximates $N_R(x_0)$ within ratio $1 + \varepsilon$. This

can be done by showing that the function $\Pi N_R(x)$ of the program variables Π and x is a 'near-invariant' of the **while** loop.

Consider a single iteration of the body of the **while** loop. Using single and double primes to denote initial and final values of the program variables we can write

$$\Pi'' = \Pi'/\alpha, \qquad N_R(x'') = \mathrm{Ext}_R(x', w).$$

Then, assuming that (4) and (5) hold, $\Pi'' N_R(x'')$ approximates $\Pi' N_R(x')$ within ratio $1+\varepsilon/2m$, which is an expression of the near-invariance property.

Since $g(x)$ diminishes on each iteration, the total number of iterations of the loop is bounded by m. On first entry into the loop, $\Pi N_R(x)$ is equal to $N_R(x_0)$, the value we wish to estimate. Immediately after the final iteration of the loop, $\Pi N_R(x)$ is just the output of the algorithm since, at that instant, $N_R(x) = 1$. These observations, combined with the near-invariance property, imply that the output of the algorithm approximates $N_R(x_0)$ within ratio $(1+\varepsilon/2m)^m$. The latter quantity is less than $1+\varepsilon$ for $\varepsilon < 1$.

The above computation was predicated on conditions (4) and (5). The proof is completed by showing that these conditions hold simultaneously with probability greater than $\frac{3}{4}$. Consider the $3t$ calls to $\mathcal{G}$ in line 1 of the algorithm. Let the r.v. T be the number of calls which yield some result. Then $E(T) = \frac{3}{2}t$, $\mathrm{Var}(T) = \frac{3}{4}t$, and hence, by Chebyshev's inequality, $\Pr(T < t) < 3/t$. Since t is bounded below (crudely) by $180m$, the probability that at least t calls are successful is greater than $1 - 1/60m$. Thus, the probability that the algorithm runs to completion (event (4)) is greater than $(1-1/60m)^m$, and hence, greater than $59/60$.

We now turn to condition (5). Consider one iteration of the **while** loop. For each $u \in \Sigma^{\sigma(x)}$, define the random variable X_u by

$$X_u = |\{y \in S : u \text{ is an initial segment of } y\}|/|S|$$

and let $\mu_u = E(X_u)$. Since X_u is an average of t independent, 0, 1 random variables, $\mathrm{Var}(X_u) \leq 1/t$ for all $u \in \Sigma^{\sigma(x)}$. Thus, by Chebyshev's inequality,

$$\begin{aligned}\Pr(|X_u - \mu_u| \leq \varepsilon/6|x|^c m) &> 1 - 36|x|^{2c} m^2/\varepsilon^2 t \\ &= 1 - 1/5|x|^c m,\end{aligned}$$

for any $u \in \Sigma^{\sigma(x)}$. Further,

$$\begin{aligned}\Pr(|\alpha - \mu_w| \leq \varepsilon/6|x|^c m) &\geq \Pr(|X_u - \mu_u| \leq \varepsilon/6|x|^c m, \forall u \in \Sigma^{\sigma(x)}) \\ &> 1 - 1/5m. \qquad (6)\end{aligned}$$

Because w is chosen to maximize $\mathrm{Ext}_R(x, w)$, α is guaranteed to be at least $|x|^{-c}$. This observation allows the *absolute* error bound on α, given by (6), to be translated to a bound on *relative* error: With probability greater than $1 - 1/5m$, α approximates μ_w within ratio $1+\varepsilon/5m$. Furthermore, since the values $y_1, \ldots, y_t$ in line 1 of the algorithm in Fig. 5 are produced by an approximate generator with tolerance $\varepsilon/11m$, we know that μ_w approximates $\mathrm{Ext}_R(x, w)/N_R(x)$ within ratio $(1+\varepsilon/11m)^2$, and

hence, within ratio $(1+\varepsilon/5m)$. Combining these two observations establishes that, with probability greater than $1-1/5m$, α approximates $\mathrm{Ext}_R(x, w)/N_R(x)$ within ratio $1+\varepsilon/2m$. Since the total number of iterations is bounded by m, the probability of event (5), *given* event (4) occurs, is at least $(1-1/5m)^m$, which is greater than or equal to $\frac{4}{5}$. Thus, the probability of events (4) and (5) occurring simultaneously is greater than $(\frac{4}{5})\times(59/60)$, which is greater than $\frac{3}{4}$, as required. □

Looking closely at the proofs of Theorems 6.3 and 6.4, it can be seen that in each case, the hypothesis of the theorem can be weakened. Let R be a self-reducible p-relation, and let the constant k_0 be such that $xRy \Rightarrow |y| = O(|x|^{k_0})$. The construction, given in Theorem 6.3, of an f.p. almost uniform generator for R, relies only on the existence of a polynomial time procedure for approximating $N_R(x)$ within ratio $(1+|x|^{-k_0})$. The existence of such a procedure implies the existence of an f.p. almost uniform generator for R, which in turn, by Theorem 6.4, implies the existence of a polynomial time algorithm for approximating $N_R(x)$ within ratio $(1+|x|^{-k})$ for any fixed k. Thus, a polynomial time counting procedure which approximates within the threshold ratio $(1+|x|^{-k_0})$ can be bootstrapped to a more accurate polynomial time counter which approximates within ratio $(1+|x|^{-k})$ for any fixed k.

Theorem 6.4 can be reviewed in a similar fashion. An almost uniform generator for R with tolerance $|x|^{-2k_0}$ can be used as a subroutine in a procedure which approximates $N_R(x)$ within ratio $(1+|x|^{-k_0})$. Hence, the existence of a polynomial time almost uniform generator for R with tolerance $|x|^{-2k_0}$, implies the existence of a polynomial time algorithm for approximating N_R within ratio $(1+|x|^{-k})$, which, as we have seen, implies the existence of an f.p. almost uniform generator for R. Thus we obtain the dramatic result that a polynomial time almost uniform generator which achieves tolerance $|x|^{-2k_0}$ can be bootstrapped to one which achieves tolerance $\exp(-|x|^k)$ for any fixed k.

Acknowledgment

The first author would like to thank Clemens Lautemann and Alistair Sinclair for discussions which helped to clarify the ideas presented in this paper.

References

[1] E. Bach, How to generate random integers with known factorisation, *Proc. 15th ACM Symp. on Theory of Computing* (1983) 184–188.

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

[3] P. Erdös and J. Spencer, *Probabilistic Methods in Combinatorics* (Academic Press, New York/London, 1974).

[4] M.R. Garey and D.S. Johnson, *Computers and Intractability* (Freeman, San Francisco, CA, 1979).

[5] J. Gill, Computational complexity of probabilistic Turing machines, *SIAM J. Comput.* 6 (1977) 675–695.

[6] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages and Computation* (Addison-Wesley, Reading, MA, 1979).
[7] R.M. Karp and M. Luby, Monte-Carlo algorithms for enumeration and reliability problems, *Proc. 24th IEEE Symp. on Foundations of Computer Science* (1983) 56-64.
[8] C.P. Schnorr, Optimal algorithms for self-reducible problems, *Proc. 3rd Internat. Coll. on Automata, Languages and Programming* (1976) 322-337.
[9] J. Simon, On the difference between one and many, *Proc. 4th Internat. Coll. on Automata, Languages and Programming* (1977) 480-491.
[10] M. Sipser, A complexity-theoretic approach to randomness, *Proc. 15th ACM Symp. on Theory of Computation* (1983) 330-335.
[11] L. Stockmeyer, The polynomial-time hierarchy, *Theoret. Comput. Sci.* **3** (1977) 1-22.
[12] L. Stockmeyer, The complexity of approximate counting, *Proc. 15th ACM Symp. on Theory of Computing* (1983) 118-126.
[13] L.G. Valiant, The complexity of combinatorial computations: An introduction, *GI 8 Jahrestagung Informatik, Fachberichte* **18** (1978) 326-337.
[14] L.G. Valiant, The complexity of computing the permanent, *Theoret. Comput. Sci.* **8** (1979) 189-201.
[15] L.G. Valiant and V.V. Vazirani, NP is as easy as detecting unique solutions, Tech. Rept. TR-14-84, Aiken Computation Laboratory, Harvard Univ., 1984.
[16] H.S. Wilf, The uniform selection of free trees, *J. Algorithms* **2** (1981) 204-207.