Texts in Computational Complexity: Counting Problems

Oded Goldreich
Department of Computer Science and Applied Mathematics
Weizmann Institute of Science, Rehovot, Israel.

January 27, 2006

We now turn to a new type of computational problems, which vastly generalize decision problems of the NP-type. We refer to counting problems, and more specifically to counting objects that can be efficiently recognized. The two formulations of NP provide a suitable definition of such objects and yield corresponding counting problems:

1. Counting the number of solutions for a given instance of a search problem (of a relation) $R \subseteq \{0,1\}^* \times \{0,1\}^*$ having efficiently checkable solutions (i.e., $R \in \mathcal{PC}$).[1] That is, on input x, we are required to output $|\{y : (x,y) \in R\}|$.

2. Counting the number of NP-witnesses (with respect to a specific verification procedure V) for a given instance of an NP-set S (i.e., $S \in \mathcal{NP}$ and V is the corresponding verification procedure). That is, on input x, we are required to output $|\{y : V(x,y) = 1\}|$.

We shall consider these counting problems as well as relaxations of them (which refer to approximating the said quantities), and see connections between these relaxed counting problems and randomized algorithms.

Contents

[1] Recall that we denote by $\mathcal{PC}$ (standing for "Polynomial-time Check") the class of search problems that correspond to polynomially-bounded binary relations that have efficiently checkable solutions. That is, $R \in \mathcal{PC}$ if the following two conditions hold:

(a) For some polynomial p, if $(x,y) \in R$ then $|y| \leq p(|x|)$.

(b) There exists a polynomial-time algorithm that given (x,y) determines whether or not $(x,y) \in R$.

1 Exact Counting

In continuation to the foregoing discussion, we define the class of problems concerned with counting efficiently recognized objects. (Recall that $\mathcal{PC}$ denotes the class of search problems having polynomially long solutions that are efficiently checkable.)

Definition 1 (counting efficiently recognized objects – $\#\mathcal{P}$): *The class $\#\mathcal{P}$ consists of all functions that count solutions to a search problem in $\mathcal{PC}$. That is, $f:\{0,1\}^* \to \mathbb{N}$ is in $\#\mathcal{P}$ if there exists $R \in \mathcal{PC}$ such that, for every x, it holds that $f(x) = |R(x)|$, where $R(x) = \{y : (x,y) \in R\}$. In this case we say that f is the* counting problem associated with *R, and denote the latter by $\#R$* (i.e., *$\#R = f$*).

Every decision problem in $\mathcal{NP}$ is Cook-reducible to $\#\mathcal{P}$, because every such problem can be cast as deciding membership in $S_R = \{x : |R(x)| > 0\}$ for some $R \in \mathcal{PC}$. It also holds that $\mathcal{BPP}$ is Cook-reducible to $\#\mathcal{P}$. The class $\#\mathcal{P}$ is sometimes defined in terms of decision problems, as is implicit in the following proposition.

Proposition 2 (a decisional version of $\#\mathcal{P}$): *For any $f \in \#\mathcal{P}$, deciding membership in $S_f \stackrel{\rm def}{=} \{(x,N) : f(x) \geq N\}$ is computationally equivalent to computing f.*

Actually, the claim holds for any function $f:\{0,1\}^* \to \mathbb{N}$ for which there exists a polynomial p such that for every $x \in \{0,1\}^*$ it holds that $f(x) \leq 2^{p(|x|)}$.

Proof: Since the relation R vouching for $f \in \#\mathcal{P}$ (i.e., $f(x) = |R(x)|$) is polynomially bounded, there exists a polynomial p such that for every x it holds that $f(x) \leq 2^{p(|x|)}$. Deciding membership in S_f is easily reduced to computing f (i.e., we accept the input (x,N) if and only if $f(x) \geq N$). Computing f is reducible to deciding S_f by using a binary search. This relies on the fact that, on input x and oracle access to S_f, we can determine whether or not $f(x) \geq N$ by making the query (x,N). Note that we know a priori that $f(x) \in [0, 2^{p(|x|)}]$. ■

The counting class $\#\mathcal{P}$ is also related to the problem of enumerating all possible solutions to a given instance (see Exercise 27).

1.1 On the power of $\#\mathcal{P}$

As indicated, $\mathcal{NP} \cup \mathcal{BPP}$ is (easily) reducible to $\#\mathcal{P}$. Furthermore, as stated in Theorem 3, the entire Polynomial-Time Hierarchy is Cook-reducible to $\#\mathcal{P}$ (i.e., $\mathcal{PH} \subseteq \mathcal{P}^{\#\mathcal{P}}$). On the other hand, any problem in $\#\mathcal{P}$ is solvable in polynomial space, and so $\mathcal{P}^{\#\mathcal{P}} \subseteq \mathcal{PSPACE}$.

Theorem 3 [14] *Every set in $\mathcal{PH}$ is Cook-reducible to $\#\mathcal{P}$.*

We do not present a proof of Theorem 3 here, because the known proofs are rather technical. Furthermore, one main idea underlying these proofs appears in a more clear form in the proof of Theorem 16.

1.2 Completeness in $\#\mathcal{P}$

The definition of $\#\mathcal{P}$-completeness is analogous to the definition of $\mathcal{NP}$-completeness. That is, *a counting problem f is* $\#\mathcal{P}$-complete *if $f \in \#\mathcal{P}$ and every problem in $\#\mathcal{P}$ is Cook-reducible to f.*

We claim that the counting problems associated with the NP-complete problems presented in previous lectures are all $\#\mathcal{P}$-complete. We warn that this fact is not due to the mere NP-completeness of these problems, but rather to an additional property of the reductions establishing their NP-completeness. Specifically, the Karp-reductions that were used (or variants of them) have the extra property of preserving the number of NP-witnesses (as captured by the following definition).

Definition 4 (parsimonious reductions): *Let $R, R' \in \mathcal{PC}$ and let g be a Karp-reduction of $S_R = \{x : R(x) \neq \emptyset\}$ to $S_{R'} = \{x : R'(x) \neq \emptyset\}$, where $R(x) = \{y : (x,y) \in R\}$ and $R'(x) = \{y : (x,y) \in R'\}$. We say that g is* parsimonious (with respect to R and R') *if for every x it holds that $|R(x)| = |R'(g(x))|$. In such a case we say that g is a* parsimonious reduction of R to R'.

We stress that the condition of being parsimonious refers to the two underlying relations R and R' (and not merely to the sets S_R and $S_{R'}$). The requirement that g is a Karp-reduction is partially redundant, because if g is polynomial-time computable and for every x it holds that $|R(x)| = |R'(g(x))|$, then g constitutes a Karp-reduction of S_R to $S_{R'}$. Specifically, $|R(x)| = |R'(g(x))|$ implies that $|R(x)| > 0$ (i.e., $x \in S_R$) if and only if $|R'(g(x))| > 0$ (i.e., $g(x) \in S_{R'}$). The reader may easily verify that the Karp-reduction underlying the proof of CSAT (and SAT) as well as many of the reductions used in the theory of NP-completeness are parsimonious.

Theorem 5 *Let $R \in \mathcal{PC}$ and suppose that every search problem in $\mathcal{PC}$ is parsimoniously reducible to R. Then the counting problem associated with R is $\#\mathcal{P}$-complete.*

Proof: Clearly, the counting problem associated with R, denoted $\#R$, is in $\#\mathcal{P}$. To show that every $f' \in \#\mathcal{P}$ is reducible to f, we consider the relation $R' \in \mathcal{PC}$ that is counted by f'; that is, $\#R' = f'$. Then, by the hypothesis, there exists a parsimonious reduction g of R' to R. This reduction also reduces $\#R'$ to $\#R$; specifically, $\#R'(x) = \#R(g(x))$ for every x. ■

Corollaries. As an immediate corollary of Theorem 5, we get that counting the number of satisfying assignments to a given CNF formula is $\#\mathcal{P}$-complete. Similar statement hold for all the other NP-complete problems mentioned in previous lectures and in fact for all NP-complete problems listed in [5]. These corollaries follow from the fact that all known reductions among natural NP-complete problems are either parsimonious or can be easily modified to be so.

We conclude that many counting problems associated with NP-complete search problems are $\#\mathcal{P}$-complete. It turns out that also counting problems associated with efficiently solvable search problems may be $\#\mathcal{P}$-complete.

Theorem 6 *There exist $\#\mathcal{P}$-complete counting problems that are associated with efficiently solvable search problems. That is, there exists $R \in \mathcal{PF}$* (i.e., R is solvable in polynomial-time) *such that $\#R$ is $\#\mathcal{P}$-complete.*

Proof: Consider the relation $R_{\mathtt{dnf}}$ consisting of pairs (ϕ, τ) such that ϕ is a DNF formula and τ is an assignment satisfying it. Note that the search problem of $R_{\mathtt{dnf}}$ is easy to solve (e.g., by picking an arbitrary truth assignment that satisfies the first term in the input formula). To see that $\#R_{\mathtt{dnf}}$ is $\#\mathcal{P}$-complete consider the following reduction from $\#R_{\mathtt{SAT}}$ (which is $\#\mathcal{P}$-complete by Theorem 5). Given a CNF formula ϕ, transform $\neg\phi$ into a DNF formula ϕ' by applying de-Morgan's Law, and return $2^n - \#R_{\mathtt{dnf}}(\phi')$, where n denotes the number of variables in ϕ (resp., ϕ'). ■

Reflections. We note that Theorem 6 is not established by a parsimonious reduction. This fact should not come as a surprise because a parsimonious reduction of $\#R'$ to $\#R$ implies that $S_{R'} = \{x : \exists y \text{ s.t. } (x,y) \in R'\}$ is reducible to $S_R = \{x : \exists y \text{ s.t. } (x,y) \in R\}$, where in our case $S_{R'}$ is NP-Complete while $S_R \in \mathcal{P}$ (since $R \in \mathcal{PF}$). Nevertheless, the proof of Theorem 6 is related to the hardness of some underlying decision problem (i.e., the problem of deciding whether a given DNF formula is a tautology (i.e., whether $\#R_{\mathtt{dnf}}(\phi') = 2^n$)). But does there exist a $\#\mathcal{P}$-complete problem that is "not based on some underlying NP-complete decision problem"? Amazingly enough, the answer is positive.

Theorem 7 [15] *Counting the number of perfect matchings in a bipartite graph is $\#\mathcal{P}$-complete.*

Equivalently (see Exercise 28), the problem of computing the permanent of matrices with 0/1-entries is $\#\mathcal{P}$-complete. Recall that the permanent of an n-by-n matrix $M = (m_{i,j})$, denoted $\mathtt{perm}(M)$, equals the sum over all permutations π of $[n]$ of the products $\prod_{i=1}^{n} m_{i,\pi(i)}$. Theorem 7 is proven by composing the following two (many-to-one) reductions (asserted in Propositions 8 and 9, respectively) and using the fact that $\#R_{\mathtt{3SAT}}$ is $\#\mathcal{P}$-complete (see Theorem 5). Needless to say, the resulting reduction is not parsimonious.

Proposition 8 *The counting problem of* 3SAT (i.e., $\#R_{\mathtt{3SAT}}$) *is reducible to computing the permanent of integer matrices. Furthermore, there exists an* even *integer $c > 0$ and a finite set of integers I such that, on input a 3CNF formula ϕ, the reduction produces an integer matrix with entries in I and a permanent value that equals $c^m \cdot \#R_{\mathtt{3SAT}}(\phi)$, where m denotes the number of clauses in ϕ.*

The original proof of Proposition 8 uses $c = 2^{10}$ and $I = \{-1, 0, 1, 2, 3\}$. It follows that, for every integer $n > 1$ that is relatively prime to c, computing the permanent modulo n is NP-hard (see Exercise 29, which also uses Theorem 16). Thus, using the case of $c = 2^{10}$, this means that computing the permanent modulo n is NP-hard for any odd $n > 1$. In contrast, computing the permanent modulo 2 (which is equivalent to computing the determinant modulo 2) is easy (i.e., can be done in polynomial-time and even in $\mathcal{NC}$). Thus, assuming $\mathcal{NP} \not\subseteq \mathcal{BPP}$, Proposition 8 cannot hold for an odd c (because by Exercise 29 it would follow that computing the permanent modulo 2 is NP-Hard). We also note that, assuming $\mathcal{P} \neq \mathcal{NP}$, Proposition 8 cannot possibly hold for a set I containing only non-negative integers (see Exercise 30).

Proposition 9 *Computing the permanent of integer matrices is reducible to computing the permanent of 0/1-matrices. Furthermore, the reduction transforms an integer matrix A into a 0/1-matrix A'' such that the permanent of A can be easily computed from A and the permanent of A''.*

The proofs of Propositions 8 and 9 are omitted.

2 Approximate Counting

Let us consider the counting problem associated with an arbitrary $R \in \mathcal{PC}$. Without loss of generality, we assume that all solutions to n-bit instances have the same length $\ell(n)$, where indeed ℓ is a polynomial. We first note that, while it may be hard to compute $\#R$, given x it is easy to approximate $\#R(x)$ up to $0.01 \cdot 2^{\ell(|x|)}$. Indeed, such an approximation is very rough, but it is not trivial. More generally, we have the following algorithm that produces an estimate of $\#R(x)$ that deviates from the correct value by an additive term that is related to the absolute bound on the number of solutions (i.e., $2^{\ell(|x|)}$).

Proposition 10 (approximation with additive deviation): *Let $R \in \mathcal{PC}$ and ℓ be a polynomial such that $R \subseteq \cup_{n\in\mathbb{N}}\{0,1\}^n \times \{0,1\}^{\ell(n)}$. Then, for every polynomial p, there exists a probabilistic polynomial-time algorithm A such that for every $x \in \{0,1\}^*$ and $\delta \in (0,1)$ it holds that*

$$\mathsf{Pr}[|A(x,\delta) - \#R(x)| > (1/p(|x|)) \cdot 2^{\ell(|x|)}] < \delta. \tag{1}$$

(As usual, δ is presented to A in binary, and hence the running time of $A(x,\delta)$ is upper-bounded by $\mathrm{poly}(|x| \cdot \log(1/\delta))$.

Proof Sketch: On input x and δ, algorithm A sets $t = \Theta(p(|x|)^2 \cdot \log(1/\delta))$, selects uniformly $y_1, ..., y_t$ and outputs $|\{i : (x, y_i) \in R\}|/t$. $\square$

Discussion. Proposition 10 is meaningful in case $\#R(x) > (1/p(|x|)) \cdot 2^{\ell(|x|)}$ holds for some x's. But otherwise, a trivial approximation (i.e., outputting the constant value zero) meets the bound of Eq. (1). In general, an approximation of $\#R(x)$ up-to a constant factor (or some other reasonable factor) is more meaningful.[2] In Section 2.1, we consider a non-trivial case where such a relative approximation can be obtained in probabilistic polynomial-time. For reasons explained in Section 2.1, we do not expect this to happen for every counting problem in $\#\mathcal{P}$, but in Section 2.2 we show that relative approximation for any problem in $\#\mathcal{P}$ can be obtained by a randomized Cook-reduction to $\mathcal{NP}$. But before turning to these results, let us state the underlying definition (and actually strengthen it by requiring approximation to within a factor of $1 \pm \varepsilon$).

Definition 11 (approximation with relative deviation): *Let $f : \{0,1\}^* \to \mathbb{N}$ and $\varepsilon, \delta : \mathbb{N} \to [0,1]$. A randomized process Π is called an* (ε, δ)-approximator of f *if for every x it holds that*

$$\mathsf{Pr}\left[|\Pi(x) - f(x)| > \varepsilon(|x|) \cdot f(x)\right] \;<\; \delta(|x|). \tag{2}$$

We say that f is efficiently $(1-\varepsilon)$-approximable (or just $(1-\varepsilon)$-approximable) *if there exists a probabilistic polynomial-time algorithm A that constitute an $(\varepsilon, 1/3)$-approximator of f.*

[2]We refrain from formally defining an F-factor approximation in this section, although we shall refer to this notion in several informal discussions. There are several ways of defining the aforementioned term (and they are all equivalent when applied to our informal discussions). For example, an F-factor approximation of $\#R$ may mean that, with high probability, the output $A(x)$ satisfies $\#R(x)/F(|x|) \leq A(x) \leq F(|x|) \cdot \#R(x)$. Alternatively, we may require that $\#R(x) \leq A(x) \leq F(|x|) \cdot \#R(x)$ (or, alternatively, that $\#R(x)/F(|x|) \leq A(x) \leq \#R(x)$.

The error probability of the latter algorithm A (which has error probability $1/3$) can be reduced to δ by $O(\log(1/\delta))$ repetitions (see Exercise 31). Typically, the running time of A will be polynomial in $1/\varepsilon$, and ε is called the deviation parameter.

2.1 Relative approximation for $\#R_{\mathtt{dnf}}$

Consider the relation $R_{\mathtt{dnf}}$ consisting of pairs (ϕ, τ) such that ϕ is a DNF formula and τ is an assignment satisfying it. Recall that the search problem of $R_{\mathtt{dnf}}$ is easy to solve and that the proof of Theorem 6 establishes that $\#R_{\mathtt{dnf}}$ is $\#\mathcal{P}$-complete (via a non-parsimonious reduction). Still there exists a probabilistic polynomial-time algorithm that provides a constant factor approximation of $\#R_{\mathtt{dnf}}$. We warn that the fact that $\#R_{\mathtt{dnf}}$ is $\#\mathcal{P}$-complete *via a non-parsimonious reduction* means that the constant factor approximation for $\#R_{\mathtt{dnf}}$ does not seem to imply a similar approximation for all problems in $\#\mathcal{P}$. In fact, we should not expect each problem in $\#\mathcal{P}$ to have a (probabilistic) polynomial-time constant-factor approximation algorithm because this would imply $\mathcal{NP} \subseteq \mathcal{BPP}$ (since a constant factor approximation allows for distinguishing the case in which the instance has no solution from the case in which the instance has a solution).

The following algorithm is actually a deterministic reduction of the task of $(\varepsilon, 1/3)$-approximating $\#R_{\mathtt{dnf}}$ to the (additive deviation) approximation provided in Proposition 10. Consider a DNF formula $\phi = \bigvee_{i=1}^{m} C_i$, where each $C_i : \{0,1\}^n \to \{0,1\}$ is a conjunction. Actually, we will deal with the more general problem in which we are (implicitly) given m subsets $S_1, ..., S_m \subseteq \{0,1\}^n$ and wish to approximate $|\bigcup_i S_i|$. In our case, each S_i is the set of assignments satisfying the conjunction C_i. In general, we make two computational assumptions regarding these sets (letting efficient mean implementable in time polynomial in $n \cdot m$):

1. Given $i \in [m]$, one can efficiently determine $|S_i|$.
2. Given $i \in [m]$ and $J \subseteq [m]$, one can efficiently approximate $\mathsf{Pr}_{s \in S_i}\left[s \in \bigcup_{j \in J} S_j\right]$ up to an additive deviation of $1/\mathrm{poly}(n+m)$.

These assumptions are satisfied in our setting (where $S_i = C_i^{-1}(1)$, see Exercise 32). The key observation towards approximating $|\bigcup_{i=1}^{m} S_i|$ is that

$$\left|\bigcup_{i=1}^{m} S_i\right| = \sum_{i=1}^{m} \left|S_i \setminus \bigcup_{j<i} S_j\right| = \sum_{i=1}^{m} |S_i| \cdot \mathsf{Pr}_{s \in S_i}\left[s \notin \bigcup_{j<i} S_j\right] \tag{3}$$

and that the probabilities in Eq. (3) can be approximated by the second assumption. This leads to the following algorithm, where ε denotes the desired deviation parameter (i.e., we wish to obtain $(1 \pm \varepsilon) \cdot |\bigcup_{i=1}^{m} S_i|$).

Construction 12 *Let $\varepsilon' = \varepsilon/m$. For $i = 1$ to m do:*

1. *Using the first assumption, compute $|S_i|$.*
2. *Using the second assumption, obtain $\widetilde{p}_i = (1 \pm \varepsilon') \cdot p_i$, where $p_i \stackrel{\text{def}}{=} \mathsf{Pr}_{s \in S_i}[s \notin \bigcup_{j<i} S_j]$. Set $a_i \stackrel{\text{def}}{=} \widetilde{p}_i \cdot |S_i|$.*

Output the sum of the a_i's.

Let $N_i = p_i \cdot |S_i|$. We are interested in the quality of the approximation to $\sum_i N_i = |\bigcup_i S_i|$ provided by $\sum_i a_i$. Using $a_i = (p_i \pm \varepsilon') \cdot |S_i| = N_i \pm \varepsilon' \cdot |S_i|$ (for all i's), we have $\sum_i a_i = \sum_i N_i \pm \varepsilon' \cdot \sum_i |S_i|$. Using $\sum_i |S_i| \leq m \cdot |\bigcup_i S_i| = m \cdot \sum_i N_i$ (and $\varepsilon = m\varepsilon'$), we get $\sum_i a_i = (1 \pm \varepsilon) \cdot \sum_i N_i$. Thus, we obtain the following result (see Exercise 32).

Proposition 13 *For every positive polynomial p, the counting problem of $R_{\mathtt{dnf}}$ is efficiently $(1-(1/p))$-approximable.*

Using the reduction presented in the proof of Theorem 6, we conclude that the number of *un*satisfying assignments to a given CNF formula is efficiently $(1-(1/p))$-approximable. We warn, however, that the number of satisfying assignments to such a formula is *not* efficiently approximable. This concurs with the general phenomenon by which *relative approximation may be possible for one quantity, but not for the complementary quantity.* Needless to say, such a phenomenon does not occur in the context of additive-deviation approximation.

2.2 Relative approximation for $\#\mathcal{P}$

Recall that we cannot expect to efficiently approximate every $\#\mathcal{P}$ problem. Specifically, efficiently approximating $\#R$ yields an efficient algorithm for deciding membership in $S_R = \{x : R(x) \neq \emptyset\}$. Thus, at best we can hope that approximating $\#R$ is not harder than deciding S_R (i.e., that approximating $\#R$ is reducible in polynomial-time to S_R). This is indeed the case for every NP-complete problem (i.e., if S_R is NP-complete). More generally, we show that approximating any problem in $\#\mathcal{P}$ is reducible in probabilistic polynomial-time to $\mathcal{NP}$.

Theorem 14 *For every $R \in \mathcal{PC}$ and positive polynomial p, there exists a probabilistic polynomial-time oracle machine that when given oracle access to $\mathcal{NP}$ constitutes a $(1/p, \mu)$-approximator of $\#R$, where μ is a negligible function* (e.g., $\mu(n) = 2^{-n}$).

Recall that it suffices to provide a $(1/p, \delta)$-approximator of $\#R$, for any constant $\delta < 0.5$, because error reduction is applicable in this context (see Exercise 31). Also, it suffices to provide a $(1/2, \delta)$-approximator for every problem in $\#\mathcal{P}$ (see Exercise 33).

Proof: Given x, we show how to approximate $|R(x)|$ to within a constant factor. The desired approximation can be obtained as in Exercise 33. We may also assume that $R(x) \neq \emptyset$, by starting with the query "is x in S_R" and halting (with output 0) if the answer is negative. Without loss of generality, we assume that $R(x) \subseteq \{0,1\}^\ell$, where $\ell = \mathrm{poly}(|x|)$. Our task is to find some $i \in \{1, ..., \ell\}$ such that $2^{i-4} \leq |R(x)| \leq 2^{i+4}$. We proceed in iterations. For $i = 1, ..., \ell+1$, we find out whether or not $|R(x)| < 2^i$. If the answer is positive then we halt with output 2^i, and otherwise we proceed to the next iteration. (Indeed, if we were able to obtain correct answers to these queries then the output 2^i would satisfy $2^{i-1} \leq |R(x)| < 2^i$.)

Needless to say, the key issue is how to check whether $|R(x)| < 2^i$. The main idea is to use a "random sieve" on the set $R(x)$ such that each element passes the sieve with probability 2^{-i}. Thus, we expect $|R(x)|/2^i$ elements of $R(x)$ to pass the sieve. Assuming that the number of elements in $R(x)$ that pass the random sieve is indeed $\lfloor |R(x)|/2^i \rfloor$, it holds that $|R(x)| \geq 2^i$ if and only if some element of $R(x)$ passes the sieve. Assuming that the sieve can be implemented efficiently, the question of whether or not some element in $R(x)$ passed the sieve is of an "NP-type" (and thus can be referred to our NP-oracle). Combining both assumptions, we may implement the foregoing process by proceeding to the next iteration as long as some element of $R(x)$ passes the sieve. Furthermore, this implementation will provide a reasonably good approximation even if the number of elements in $R(x)$ that pass the random sieve is only approximately equal to $|R(x)|/2^i$. In fact, the level of approximation that this implementation provides is closely related to the level of approximation that is provided by the random sieve. Details follow.

Implementing a random sieve. The random sieve is implemented by using a family of hashing functions (see Appendix). Specifically, in the i^{th} iteration we use a family H_ℓ^i such that each $h \in H_\ell^i$ has a poly(ℓ)-bit long description and maps ℓ-bit long strings to i-bit long strings. Furthermore, the family is accompanied with an efficient evaluation algorithm (i.e., mapping adequate pairs (h,x) to $h(x)$) and satisfies (for every $S \subseteq \{0,1\}^\ell$)

$$\mathsf{Pr}_{h\in H_\ell^i}[|\{y \in S : h(y) = 0^i\}| \notin (1 \pm \varepsilon) \cdot 2^{-i}|S|] \;<\; \frac{2^i}{\varepsilon^2|S|} \tag{4}$$

(see Lemma 24). The random sieve will let y pass if and only if $h(y) = 0^i$. Indeed, this random sieve is not as perfect as we assumed in the foregoing discussion, but Eq. (4) says that in some sense this sieve is good enough.

Implementing the queries. Recall that for some x, i and $h \in H_\ell^i$, we need to determine whether $\{y \in R(x) : h(y) = 0^i\} = \emptyset$. This type of question can be cast as membership in the set

$$S_{R,H} \stackrel{\text{def}}{=} \{(x,i,h) : \exists y \text{ s.t. } (x,y) \in R \wedge h(y) = 0^i\}. \tag{5}$$

Using the hypotheses that $R \in \mathcal{PC}$ and that the family of hashing functions has an efficient evaluation algorithm, it follows that $S_{R,H}$ is in $\mathcal{NP}$.

The actual procedure. On input $x \in S_R$ and oracle access to $S_{R,H}$, we proceed in iterations, starting with $i = 1$ and halting at $i = \ell$ (if not before), where ℓ denotes the length of the potential solutions for x. In the i^{th} iteration (where $i < \ell$), we uniformly select $h \in H_\ell^i$ and query the oracle on whether or not $(x,i,h) \in S_{R,H}$. If the answer is negative then we halt with output 2^i, and otherwise we proceed to the next iteration (using $i \leftarrow i+1$). Needless to say, if we reach the last iteration (i.e., $i = \ell$) then we just halt with output 2^ℓ.

Indeed, we have ignored the case that $x \notin S_R$, which can be easily handled by a minor modification of the foregoing procedure. Specifically, on input x, we first query S_R on x and halt with output 0 if the answer is negative. Otherwise we proceed as in the foregoing procedure.

The analysis. We upper-bound separately the probability that the procedure outputs a value that is too small and the probability that it outputs a value that is too big. In light of the foregoing discussion, we may assume that $|R(x)| > 0$, and let $i_x = \lfloor \log_2 |R(x)| \rfloor \geq 0$.

1. The probability that the procedure *halts in a specific iteration* $i < i_x$ equals $\mathsf{Pr}_{h\in H_\ell^i}[|\{y \in R(x) : h(y) = 0^i\}| = 0]$, which in turn is upper-bounded by $2^i/|R(x)|$ (using Eq. (4) with $\varepsilon = 1$). Thus, the probability that the procedure halts *before* iteration $i_x - 3$ is upper-bounded by $\sum_{i=0}^{i_x-4} 2^i/|R(x)|$, which in turn is less than $1/8$ (because $i_x \leq \log_2 |R(x)|$). Thus, with probability at least $7/8$, the output is at least $2^{i_x-3} > |R(x)|/16$ (because $i_x > (\log_2 |R(x)|) - 1$).

2. The probability that the procedure *does not halt in iteration* $i > i_x$ equals $\mathsf{Pr}_{h\in H_\ell^i}[|\{y \in R(x) : h(y) = 0^i\}| \geq 1]$, which in turn is upper-bounded by $\alpha/(\alpha-1)^2$, where $\alpha = 2^i/|R(x)| > 1$ (using Eq. (4) with $\varepsilon = \alpha - 1$).[3] Thus, the probability that the procedure does not halt by

[3] A better bound can be obtained by using the hypothesis that, for every y, when h is uniformly selected in H_ℓ^i, the value of $h(y)$ is uniformly distributed in $\{0,1\}^i$. In this case, $\mathsf{Pr}_{h\in H_\ell^i}[|\{y \in R(x) : h(y) = 0^i\}| \geq 1]$ is upper-bounded by $\mathsf{E}_{h\in H_\ell^i}[|\{y \in R(x) : h(y) = 0^i\}|] = |R(x)|/2^i$.

iteration $i_x + 4$ is upper-bounded by $8/49 < 1/6$ (because $i_x > (\log_2 |R(x)|) - 1$). Thus, with probability at least $5/6$, the output is at most $2^{i_x+4} \leq 16 \cdot |R(x)|$ (because $i_x \leq \log_2 |R(x)|$).

Thus, with probability at least $(7/8) - (1/6) > 2/3$, the foregoing procedure outputs a value v such that $v/16 \leq |R(x)| < 16v$. Reducing the deviation by using the ideas presented in Exercise 33 (and reducing the error probability as in Exercise 31), the theorem follows. ■

Perspective. The key observation underlying the proof Theorem 14 is that while we cannot test (even with the help of an NP-oracle) whether the number of solutions is greater than a given number, we can test (with the help of an NP-oracle) whether the number of solutions that "survive a random sieve" is greater than zero. If fact, we can also test whether the number of solutions that "survive a random sieve" is greater than a small number, where small means polynomial in the length of the input (see Exercise 35). In general, our complexity is linear in the size of the threshold, and not in the length of its binary description. Indeed, in many settings it is more advantageous to use a threshold that is polynomial in some efficiency parameter (rather than using the threshold zero); examples appear in Section 4.2 and in [7].

3 Searching for unique solutions

A natural computational problem (regarding search problems), which arises when discussing the number of solutions, is the problem of distinguishing instances having a single solution from instances having no solution (or finding the unique solution whenever such exists). We mention that instances having a single solution facilitate numerous arguments (see, for example, Exercise 29 and [2]). Formally, searching for and deciding the existence of unique solutions are defined within the framework of promise problems.

Definition 15 (search and decision problems for unique solution instances): *The* set of instances having unique solutions *with respect to the binary relation R is defined as* $\mathtt{US}_R \stackrel{\text{def}}{=} \{x : |R(x)| = 1\}$, *where* $R(x) \stackrel{\text{def}}{=} \{y : (x,y) \in R\}$*. As usual, we denote* $S_R = \{x : |R(x)| \geq 1\}$*, and* $\overline{S}_R \stackrel{\text{def}}{=} \{0,1\}^* \setminus S_R = \{x : |R(x)| = 0\}$.

- *The problem of* finding unique solutions for R *is defined as the* search problem R with promise $\mathtt{US}_R \cup \overline{S}_R$.[4]

[4] A search problem with a promise consists of a binary relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ and a promise set P. Such a problem is also referred to as the search problem R with promise P.

- The search problem R with promise P is solved by algorithm A if for every $x \in P$ it holds that $(x, A(x)) \in R$ if $x \in S_R = \{x : R(x) \neq \emptyset\}$ and $A(x) = \perp$ otherwise, where $R(x) = \{y : (x,y) \in R\}$.

  The time complexity of A on inputs in P is defined as $T_{A|P}(n) \stackrel{\text{def}}{=} \max_{x \in P \cap \{0,1\}^n} \{t_A(x)\}$, where $t_A(x)$ is the running time of $A(x)$ and $T_{A|P}(n) = 0$ if $P \cap \{0,1\}^n = \emptyset$.
- The search problem R with promise P is in the promise problem extension of $\mathcal{PF}$ if there exists a polynomial-time algorithm that solves this problem.
- The search problem R with promise P is in the promise problem extension of $\mathcal{PC}$ if there exists a polynomial T and an algorithm A such that, for every $x \in P$ and $y \in \{0,1\}^*$, algorithm A makes at most $T(|x|)$ steps and it holds that $A(x,y) = 1$ if and only if $(x,y) \in R$.

An algorithm A solves the candid search problem of the binary relation R if for every $x \in S_R \stackrel{\text{def}}{=} \{x : \exists y \text{ s.t. } (x,y) \in R\}$ it holds that $(x, A(x)) \in R$. The time complexity of such an algorithm is defined as $T_{A|S_R}(n) \stackrel{\text{def}}{=} \max_{x \in P \cap \{0,1\}^n} \{t_A(x)\}$, where $t_A(x)$ is the running time of $A(x)$ and $T_{A|S_R}(n) = 0$ if $P \cap \{0,1\}^n = \emptyset$.

In continuation to the notion of candid search problems, the candid searching for unique solutions for R *is defined as the* search problem R with promise $\mathtt{US}_R$.

- *The problem of* deciding unique solution for R *is defined as the* promise problem $(\mathtt{US}_R, \overline{S}_R)$.

Interestingly, in many natural cases, the promise does not make any of these problems any easier than the original problem. That is, for all known NP-complete problems, the original problem is reducible in probabilistic polynomial-time to the corresponding unique instances problem.

Theorem 16 *Let $R \in \mathcal{PC}$ and suppose that every search problem in $\mathcal{PC}$ is parsimoniously reducible to R. Then solving the search problem of R* (resp., deciding membership in S_R) *is reducible in probabilistic polynomial-time to finding unique solutions for R* (resp., to the promise problem $(\mathtt{US}_R, \overline{S}_R)$). *Furthermore, there exists a probabilistic polynomial-time computable mapping M such that for every $x \in \overline{S}_R$ it holds that $M(x) \in \overline{S}_R$, whereas for every $x \in S_R$ it holds that $\Pr[M(x) \in \mathtt{US}_R] \geq 1/\mathrm{poly}(|x|)$.*

We note that the condition regarding parsimonious reductions is crucial (see Exercise 36).

Proof: As in the proof of Theorem 14, the idea is to apply a "random sieve" on $R(x)$, this time with the hope that a single element survives. Specifically, if we let each element passes the sieve with probability approximately $1/|R(x)|$ then with constant probability a single element survives. Sieving will be performed by a random function selected in an adequate hashing family (see Appendix). A couple of questions arise:

1. *How do we get an approximation to $|R(x)|$?* Note that we need such an approximation in order to determine the adequate hashing family. Indeed, we may just invoke Theorem 14, but this will not yield a many-to-one reduction. Instead, we just select $m \in \{0, ..., \mathrm{poly}(|x|)\}$ uniformly and note that (if $|R(x)| > 0$ then) $\Pr[m = \lceil \log_2 |R(x)| \rceil] = 1/\mathrm{poly}(|x|)$.

   Thus, we randomly map x to (x, m, h), where h is uniformly selected in an adequate hashing family.

2. *How does the question of whether a single element of $R(x)$ pass the random sieve translate to an instance of the unique-instance problem for R?* Recall that in the proof of Theorem 14 the non-emptiness of the set of element of $R(x)$ that pass the sieve defined by h was determined by checking membership (of (x, m, h)) in $S_{R,H} \in \mathcal{NP}$ (defined in Eq. (5)). Furthermore, the number of NP-witnesses for $(x, m, h) \in S_{R,H}$ equals the number of elements of $R(x)$ that pass the sieve. Using the parsimonious reduction of $S_{R,H}$ to S_R (which is guaranteed by the theorem's hypothesis), we obtained the desired instance.

Note that in case $R(x) = \emptyset$ the aforementioned mapping always generates a no-instance (of $S_{R,H}$ and thus of S_R). Details follow.

Implementation (i.e., the mapping M). As in the proof of Theorem 14, we assume, without loss of generality, that $R(x) \subseteq \{0,1\}^\ell$, where $\ell = \mathrm{poly}(|x|)$. We start by uniformly selecting $m \in \{0, 1, ..., \ell\}$ and $h \in H_\ell^m$, where H_ℓ^m is a family of efficiently computable and pairwise-independent hashing functions (see Definition 21) mapping ℓ-bit long strings to m-bit long strings.[5] Thus, we obtain an instance (x, m, h) of $S_{R,H} \in \mathcal{NP}$ such that the set of valid solutions for (x, m, h) equals $\{y \in R(x) : h(y) = 0^m\}$. Using the parsimonious reduction g of $S_{R,H}$ to S_R, we map (x, m, h) to

[5]For sake of uniformity, we allow also the case of $m = 0$, which is rather artificial. In this case all hashing functions in H_ℓ^0 map $\{0,1\}^\ell$ to the empty string, which is viewed as 0^0.

$g(x,m,h)$, and it holds that $|\{y \in R(x) : h(y)=0^m\}|$ equals $|R(g(x,m,h))|$. To summarize, on input x the randomized mapping M outputs the instance $M(x) \stackrel{\rm def}{=} g(x,m,h)$, where $m \in \{0,1,...,\ell\}$ and $h \in H_\ell^m$ are uniformly selected.

The analysis. Note that for any $x \in \overline{S}_R$ it holds that $\Pr[M(x) \in \overline{S}_R] = 1$. Assuming that $x \in S_R$, with probability exactly $1/(\ell+1)$ it holds that $m = m_x$, where $m_x \stackrel{\rm def}{=} \lceil \log_2 |R(x)| \rceil$. In this case, for a uniformly selected $h \in H_\ell^m$, we lower-bound the probability that $\{y \in R(x) : h(y)=0^m\}$ is a singleton. Using the Inclusion-Exclusion Principle, we have

$$\begin{aligned} & \Pr_{h \in H_\ell^{m_x}}[|\{y \in R(x) : h(y)=0^{m_x}\}| = 1] \qquad (6) \\ \geq & \sum_{y \in R(x)} \Pr_{h \in H_\ell^{m_x}}[h(y)=0^{m_x}] \\ & - \sum_{y_1 < y_2 \in R(x)} \Pr_{h \in H_\ell^{m_x}}[h(y_1)=h(y_2)=0^{m_x}] \\ = & |R(x)| \cdot 2^{-m_x} - \binom{|R(x)|}{2} \cdot 2^{-2m_x} \end{aligned}$$

where the equality is due to the pairwise independence property. Using $2^{m_x-1} < |R(x)| \leq 2^{m_x}$, it follows that Eq. (6) is lower-bounded by 1/4. Thus, $\Pr[M(x) \in \mathtt{US}_R] \geq 1/4(\ell+1)$, and the theorem follows. ∎

Comment. Theorem 16 is sometimes stated as referring to the unique solution problem of SAT. In this case and when using a specific family of pairwise independent hashing functions, the use of the parsimonious reduction can be avoided. For details see Exercise 37.

4 Uniform generation of solutions

We now turn to a new type of computational problems, which may be viewed as a straining of search problems. We refer to the task of generating a uniformly distributed solution for a given instance, rather than merely finding an adequate solution. Needless to say, by definition, algorithms solving this ("uniform generation") task must be randomized. Focusing on relations in $\mathcal{PC}$ we consider two versions of the problem, which differ by the level of approximation provided for the desired (uniform) distribution.[6]

Definition 17 (uniform generation): *Let $R \in \mathcal{PC}$ and $S_R = \{x : |R(x)| \geq 1\}$, and let Π be a probabilistic process.*

1. *We say that Π* solves the uniform generation problem *of R if, on input $x \in S_R$, the process Π outputs either an element of $R(x)$ or a special symbol, denoted $\perp$, such that $\Pr[\Pi(x) \in R(x)] \geq 1/2$ and for every $y \in R(x)$ it holds that $\Pr[\Pi(x)=y \mid \Pi(x) \in R(x)] = 1/|R(x)|$.*
2. *For $\varepsilon : \mathbb{N} \to [0,1]$, we say that Π* solves the $(1-\varepsilon)$-approximate uniform generation problem of *R if, on input $x \in S_R$, the distribution $\Pi(x)$ is $\varepsilon(|x|)$-close to the uniform distribution on $R(x)$.*

[6]Note that a probabilistic algorithm running in strict polynomial-time is not able to output a perfectly uniform distribution on sets of certain sizes. Specifically, referring to the standard model that allows only for uniformly selected binary values, such algorithms cannot output a perfectly uniform distribution on sets having cardinality that is not a power of two.

In both cases, without loss of generality, we may require that if $x \notin S_R$ then $\Pr[\Pi(x) = \perp] = 1$. *More generally, we may require that* Π *never outputs a string not in* $R(x)$.

Note that the error probability of uniform generation (as in Item 1) can be made exponentially vanishing (in $|x|$) by employing error-reduction. In contrast, we are not aware of any general way of reducing the deviation of an approximate uniform generation procedure (as in Item 2).[7]

In Section 4.1 we show that, for many search problems, approximate uniform generation is computationally equivalent to approximate counting. In Section 4.2 we present a direct approach for solving the uniform generation problem of any search problem in $\mathcal{PC}$ by using an oracle to $\mathcal{NP}$.

4.1 Relation to approximate counting

We show that for every $R \in \mathcal{PC}$ that is NP-complete under parsimonious reductions, the approximate counting problem associated with R is computationally equivalent to approximate uniform generation with respect to R. Recalling that both approximate problems are parameterized by the level of precision, we obtain the following quantitative form of the aforementioned equivalence.

Theorem 18 *Let $R \in \mathcal{PC}$ and let ℓ be a polynomial such that for every $(x, y) \in R$ it holds that $|y| \leq \ell(|x|)$. Suppose that every search problem in $\mathcal{PC}$ is parsimoniously reducible to R.*

1. From approximate counting to approximate uniform generation: *Let $\varepsilon(n) = 1/5\ell(n)$ and let $\mu : \mathbb{N} \to (0, 1)$ be a function satisfying $\mu(n) \geq \exp(-\mathrm{poly}(n))$. Then, $(1 - \mu)$-approximate uniform generation for R is reducible in probabilistic polynomial-time to $(1-\varepsilon)$-approximating $\#R$.*

2. From approximate uniform generation to approximate counting: *For every noticeable ε : $\mathbb{N} \to (0, 1)$* (i.e., $\varepsilon(n) \geq 1/\mathrm{poly}(n)$ for every n)*, the problem of $(1 - \varepsilon)$-approximating $\#R$ is reducible in probabilistic polynomial-time to $(1 - \varepsilon')$-approximate uniform generation problem of R, where $\varepsilon'(n) = \varepsilon(n)/5\ell(n)$.*

Note that the quality of the approximate uniform generation asserted in Part 1 (i.e., μ) is independent of the quality of the approximate counting procedure (i.e., ε) to which the former is reduced, provided that the approximate counter performs better than some threshold. On the other hand, the quality of the approximate counting asserted in Part 2 (i.e., ε) does depend on the quality of the approximate uniform generation (i.e., ε'). Recall, however, that the quality of approximate counting procedures for problems that are NP-complete under parsimonious reductions can be improved (see Exercise 34). Thus, for such problems, the quality of approximate uniform generation procedures can be improved by applying both parts of Theorem 18.

Proof: Throughout the proof, we assume for simplicity (and in fact without loss of generality) that $R(x) \neq \emptyset$ and $R(x) \subseteq \{0, 1\}^{\ell(|x|)}$.

Towards Part 1, let use first reduce the uniform generation problem of R to $\#R$ (rather than to approximating $\#R$). On input $x \in S_R$, we generate a uniformly distributed $y \in R(x)$ by randomly generating its bits one after the other. We proceed in iterations, entering the i^{th} iteration with an $(i-1)$-bit long string y' such that $R'(x; y') \stackrel{\text{def}}{=} \{y'' : (x, y'y'') \in R\}$ is not empty. With probability $|R'(x; y'1)|/|R'(x; y')|$ we set the i^{th} bit to equal 1, and otherwise we set it to equal 0. We obtain both $|R'(x; y'1)|$ and $|R'(x; y')|$ by using a parsimonious reduction g of $R' = \{((x; y'), y'') : (x, y'y'') \in$

[7] We note that in some cases, the deviation of an approximate uniform generation procedure can be reduced. See discussion following Theorem 18.

$R\} \in \mathcal{PC}$ to R. That is, we obtain $|R'(x;y')|$ by querying for the value of $|R(g(x;y'))|$. Ignoring integrality issues, all this works perfectly (i.e., we generate an $\ell(n)$-bit string uniformly distributed in $R(x)$) as long as we have oracle access to $\#R$. But we only have oracle access to an approximation of $\#R$, and thus a careful modification is in place.

Let us denote the approximation oracle by A. Firstly, by adequate error reduction, we may assume that, for every x, it holds that $\mathsf{Pr}[A(x) \in (1 \pm \varepsilon(n)) \cdot \#R(x)] > 1 - \mu'(|x|)$, where $\mu'(n) = \mu(n)/\ell(n)$. In the rest of the analysis we ignore the probability that the estimate deviates from the aforementioned interval, and note that this rare event is the only source of the possible deviation of the output distribution from the uniform distribution on $R(x)$.[8] Let us assume for a moment that A is *deterministic* and that for every x and y' it holds that

$$A(g(x,y'0)) + A(g(x,y'1)) \;\le\; A(g(x;y')). \tag{7}$$

We also assume that the approximation is correct at the "trivial level" (where one may just check whether or not (x,y) is in R); that is, for every $y \in \{0,1\}^{\ell(|x|)}$, it holds that

$$A(g(x;y)) = 1 \text{ if } (x,y) \in R \text{ and } A(g(x;y)) = 0 \text{ otherwise.} \tag{8}$$

We modify the i^{th} iteration of the foregoing procedure such that, when entering with the $(i-1)$-bit long prefix y', we set the i^{th} bit to 1 (resp., to 0) with probability $A(g(x;y'1))/A(g(x;y'))$ (resp., with probability $A(g(x;y'0))/A(g(x;y')))$ and halt (with output $\perp$) with the residual probability. If we completed the last (i.e., $\ell(|x|)^{\text{th}}$) iteration, then we output the $\ell(|x|)$-bit long string that was generated. Thus, as long as Eq. (7) holds (but regardless of other aspects of the quality of the approximation), every $y = \sigma_1 \cdots \sigma_{\ell(|x|)} \in R(x)$, is output with probability

$$\frac{A(g(x;\sigma_1))}{A(g(x;\lambda))} \cdot \frac{A(g(x;\sigma_1\sigma_2))}{A(g(x;\sigma_1))} \cdots \frac{A(g(x;\sigma_1\sigma_2\cdots\sigma_{\ell(|x|)}))}{A(g(x;\sigma_1\sigma_2\cdots\sigma_{\ell(|x|)-1}))} \tag{9}$$

which, by Eq. (8), equals $1/A(g(x;\lambda))$. Thus, the procedure outputs each element of $R(x)$ with equal probability, and never outputs a non-$\perp$ value that is outside $R(x)$. It follows that the quality of approximation only effects the probability that the procedure outputs a non-$\perp$ value (which equals $|R(x)|/A(g(x;\lambda))$).

We now turn to enforcing Eq. (7) and Eq. (8). We may enforce Eq. (8) by performing the straightforward check (of whether or not $(x,y) \in R$) rather than invoking $A(g(x,y))$.[9] As for Eq. (7), we enforce it artificially by using $A'(x,y') \stackrel{\text{def}}{=} (1+\varepsilon(|x|))^{3(\ell(|x|)-|y'|)} \cdot A(g(x;y'))$ instead of $A(g(x;y'))$. Recalling that $A(g(x;y')) = (1 \pm \varepsilon(|xy'|)) \cdot |R'(x;y')|$, we have

$$\begin{aligned} A'(x,y') &> (1+\varepsilon(|x|))^{3(\ell(|x|)-|y'|)} \cdot (1-\varepsilon(|x|)) \cdot |R'(x;y')| \\ A'(x,y'\sigma) &< (1+\varepsilon(|x|))^{3(\ell(|x|)-|y'|-1)} \cdot (1+\varepsilon(|x|)) \cdot |R'(x;y'\sigma)| \end{aligned}$$

and the claim follows using $(1-\varepsilon(|x|))\cdot(1+\varepsilon(|x|))^3 > (1-\varepsilon(|x|))$. Note that the foregoing modification only decreases the probability of outputting a non-$\perp$ value by a factor of $(1+\varepsilon(|x|))^{3\ell(|x|)} < 2$, where the inequality is due to the setting of ε (i.e., $\varepsilon(n) = 1/5\ell(n)$). Finally, we refer to our assumption that A is deterministic. This assumption was only used in order to identify the value of $A(g(x,y'))$ obtained and used in the $(|y'|-1)^{\text{st}}$ iteration with the value of $A(g(x,y'))$ obtained

[8] The possible deviation is due to the fact that this rare event may occur with different probability in the different invocations of algorithm A.

[9] Alternatively, we note that since A is a $(1-\varepsilon)$-approximator for $\varepsilon < 1$ it must hold that $\#R'(z) = 0$ implies $A(z) = 0$. Also, since $\varepsilon < 1/3$, if $\#R'(z) = 1$ then $A(z) \in (2/3, 4/3)$, which may be rounded to 1.

and used in the $|y'|^{\text{th}}$ iteration, but the same effect can be obtained by just using the former value (in the $|y'|^{\text{th}}$ iteration) rather than re-invoking A in order to obtain it. Part 1 follows.

Towards Part 2, let use first reduce the task of approximating $\#R$ to the task of (exact) uniform generation for R. On input $x \in S_R$, the reduction uses the tree of possible prefixes of elements of $R(x)$ in a somewhat different manner. Again, we proceed in iterations, entering the i^{th} iteration with an $(i-1)$-bit long string y' such that $R'(x;y') \stackrel{\text{def}}{=} \{y'' : (x, y'y'') \in R\}$ is not empty. At the i^{th} iteration we estimate the bigger among the two fractions $|R'(x;y'0)|/|R'(x;y')|$ and $|R'(x;y'1)|/|R'(x;y')|$, by uniformly sampling the uniform distribution over $R'(x;y')$. That is, taking $\text{poly}(|x|/\varepsilon'(|x|))$ uniformly distributed samples in $R'(x;y')$, we obtain with overwhelmingly high probability an approximation of these fractions up to an additive deviation of at most $\varepsilon'(|x|)/3$. This means that we obtain a relative approximation up-to a factor of $1 \pm \varepsilon'(|x|)$ for the fraction (or fractions) that is (resp., are) bigger than 1/3. Indeed, we may not be able to obtain such a good relative approximation of the other fraction (in case it is very small), but this does not matter. It also does not matter that we cannot tell which is the bigger fraction among the two; it only matter that we use an approximation that indicates a quantity that is, say, bigger than 1/3. We proceed to the next iteration by augmenting y' using the bit that corresponds to such a quantity. Specifically, suppose that we obtained the approximations $a_0(y') \approx |R'(x;y'0)|/|R'(x;y')|$ and $a_1(y') \approx |R'(x;y'1)|/|R'(x;y')|$. Then we extend y' by the bit 1 if $a_1(y') > a_0(y')$ and extend y' by the bit 0 otherwise. Finally, when we reach $y = \sigma_1 \cdots \sigma_{\ell(|x|)}$ such that $(x,y) \in R$, we output

$$a_{\sigma_1}(\lambda)^{-1} \cdot a_{\sigma_2}(\sigma_1)^{-1} \cdots a_{\sigma_{\ell(|x|)}}(\sigma_1\sigma_2 \cdots \sigma_{\ell(|x|)-1})^{-1}. \tag{10}$$

As in Part 1, actions regarding R' (in this case uniform generation in R') are conducted via the parsimonious reduction g to R. That is, whenever we need to sample uniformly in the set $R'(x;y')$, we sample the set $R(g(x;y'))$ instead. Finally, note that the deviation from uniform distribution (i.e., the fact that we can only approximately sample R) merely introduces such a deviation in each of our approximations to the relevant fractions (i.e., to a fraction bigger than 1/3). Specifically, on input x, using an oracle that provides a $(1-\varepsilon')$-approximate uniform generation for R, with overwhelmingly high probability, the output (as defined in Eq. (10)) is in

$$\prod_{i=1}^{\ell(|x|)} \left((1 \pm 2\varepsilon'(|x|)) \cdot \frac{|R'(x;\sigma_1 \cdots \sigma_{i-1})|}{|R'(x;\sigma_1 \cdots \sigma_i)|} \right) \tag{11}$$

where the error probability is due to the unlikely case that in one of the iterations our approximations deviates from the correct value by more than an additive deviation term of $\varepsilon'(n)/3$. Noting that Eq. (11) equals $(1 \pm 2\varepsilon'(|x|))^{\ell(|x|)} \cdot |R(x)|$ and using $(1 \pm 2\varepsilon'(|x|))^{\ell(|x|)} \subset (1 \pm \varepsilon(|x|))$, Part 2 follows, and so does the theorem. ■

4.2 Direct uniform generation

We conclude the current section by presenting a direct procedure for solving the uniform generation problem of any $R \in \mathcal{PC}$. This procedure uses an oracle to $\mathcal{NP}$, which is unavoidable because solving the uniform generation problem implies solving the corresponding search problem. One advantage of this process, over the reduction presented in Section 4.1, is that it solves the uniform generation problem rather than the *approximate* uniform generation problem.

We are going to use hashing again, but this time we use a family of hashing functions having a stronger "uniformity property" (see Section A.3). Specifically, we will use a family of ℓ-wise independent hashing functions mapping ℓ-bit strings to m-bit strings, where ℓ bounds the length of solutions in R, and rely on the fact that such a family satisfies Lemma 26. Intuitively, such functions partition $\{0,1\}^\ell$ into 2^m cells and Lemma 26 asserts that these partitions "uniformly shatter" all sufficiently large sets. That is, for every set $S \subseteq \{0,1\}^\ell$ of size $\Omega(\ell \cdot 2^m)$ the partition induced by almost every function is such that each cell contains approximately $|S|/2^m$ elements of S. In particular, if $|S| = \Theta(\ell \cdot 2^m)$ then each cell contains $\Theta(\ell)$ elements of S.

In the following construction, we assume that on input x we also obtain a good approximation to the size of $R(x)$. This assumption can be enforced by using an approximate counting procedure as a preprocessing stage. Alternatively, the ideas presented in the following construction yield such an approximate counting procedure.

Construction 19 (uniform generation): *On input x and $m'_x \in \{m_x, m_x+1\}$, where $m_x \stackrel{\rm def}{=} \lfloor \log_2 |R(x)| \rfloor$ and $R(x) \subseteq \{0,1\}^\ell$, the oracle machine proceeds as follows.*

1. Selecting a partition that "uniformly shatters" $R(x)$. *The machine sets $m = \max(0, m'_x - 6 - \log_2 \ell)$ and selects uniformly $h \in H_\ell^m$. Such a function defines a partition of $\{0,1\}^\ell$ into 2^m cells*[10]*, and the hope is that each cell contains approximately the same elements of $R(x)$. Next, the machine checks that this is indeed the case or rather than no cell contains more that 1000ℓ elements of $R(x)$. This is done by checking whether or not $(x,h,1^{1000\ell})$ is in the set $S^{(1)}_{R,H}$ defined as follows*

$$\begin{aligned} S^{(1)}_{R,H} &\stackrel{\rm def}{=} \{(x',h',1^t) : \exists v \text{ s.t. } |\{y : (x',y) \in R \wedge h'(y) = v\}| \geq t\} \\ &= \{(x',h',1^t) : \exists v, y_1, ..., y_t \text{ s.t. } \psi^{(1)}(x',h',v,y_1,...,y_t)\}, \end{aligned} \tag{12}$$

where $\psi^{(1)}(x',h',v,y_1,...,y_t)$ holds if and only if $y_1 < y_2 \cdots < y_t$ and for every $j \in [t]$ it holds that $(x',y_j) \in R \wedge h'(y_j) = v$. Note that $S^{(1)}_{R,H} \in \mathcal{NP}$.

   If the answer is positive (i.e., there exists a cell that contains more that 1000ℓ elements of $R(x)$) *then the machine halts with output $\perp$. Otherwise, the machine continues with this choice of h. In this case, for every $v \in \{0,1\}^m$, it holds that no cell contains more that 1000ℓ elements of $R(x)$* (i.e., $|\{y : (x,y) \in R \wedge h(y) = v\}| < 1000\ell$)*. We stress that this is an absolute guarantee that follows from $(x,h,1^{1000\ell}) \notin S^{(1)}_{R,H}$.*

2. Selecting a cell and determining the number of elements of $R(x)$ that are contained in it. *The machine selects uniformly $v \in \{0,1\}^m$ and determines $s_v \stackrel{\rm def}{=} |\{y : (x,y) \in R \wedge h(y) = v\}|$ by making queries to the following NP-set*

$$S^{(2)}_{R,H} \stackrel{\rm def}{=} \{(x',h',v',1^t) : \exists y_1, ..., y_t \text{ s.t. } \psi^{(1)}(x',h',v',y_1,...,y_t)\}. \tag{13}$$

   Specifically, for $i = 1, ..., 1000\ell$, it checks whether $(x,h,v,1^i)$ is in $S^{(2)}_{R,H}$, and sets s_v to be the largest value of i for which the answer is positive.

3. Obtaining all the elements of $R(x)$ that are contained in the selected cell, and outputting one of them at random. *Using s_v, the procedure reconstructs the set $S_v \stackrel{\rm def}{=} \{y : (x,y) \in R \wedge h(y) = v\}$, by making queries to the following NP-set*

$$S^{(3)}_{R,H} \stackrel{\rm def}{=} \{(x',h',v',1^t,j) : \exists y_1, ..., y_t \text{ s.t. } \psi^{(3)}(x',h',v',y_1,...,y_t,j)\}, \tag{14}$$

[10] For sake of uniformity, we allow also the case of $m = 0$, which is rather artificial. In this case all hashing functions in H_ℓ^0 map $\{0,1\}^\ell$ to the empty string, which is viewed as 0^0, and thus define a trivial partition of $\{0,1\}^\ell$ (i.e., into a single cell).

where $\psi^{(3)}(x', h', v', y_1, ..., y_t, j)$ holds if and only if $\psi^{(1)}(x', h', v', y_1, ..., y_t)$ holds and the j^{th} bit of $y_1 \cdots y_t$ equals 1. Specifically, for $j_1 = 1, ..., s_v$ and $j_2 = 1, ..., \ell$, we make the query $(x, h, v, 1^{s_v}, (j_1 - 1) \cdot \ell + j_2)$ in order to determine the j_2^{th} bit of y_{j_1}. Finally, having recovered S_v, the procedure outputs each $y \in S_v$ with probability $1/1000\ell$, and outputs $\perp$ otherwise.

By Lemma 26 (and $m \leq m_x + 1 - 6 - \log_2 \ell$), with overwhelmingly high probability, each set $\{y : (x, y) \in R \wedge h(y) = v\}$ has cardinality $(1 \pm 0.5)|R(x)|/2^m$. Using $m'_x > (\log_2 |R(x)|) - 1$ (resp., $m'_x \leq (\log_2 |R(x)|) + 1$), it follows that $|R(x)|/2^m < 1000\ell$ (resp., $|R(x)|/2^m > 10\ell$). Thus, Step 1 can be easily adapted to yield an approximate counting procedure for $\#R$ (see Exercise 38). However, our aim is to establish the following fact.

Proposition 20 *Construction 19 solves the uniform generation problem of R.*

Proof: By Lemma 26 (and the setting of m), with overwhelmingly high probability, a uniformly selected $h \in H_\ell^m$ partitions $R(x)$ into 2^m cells, each containing at most 1000ℓ elements. The key observation, stated in Step 1, is that if the procedure does not halt in Step 1 then it is indeed the case that h induces such a partition. The fact that these cells may contain a different number of elements is immaterial, because each element is output with the same probability (i.e., $1/1000\ell$). What matters is that the average number of elements in the cells is sufficiently large, because this average number determines the probability that the procedure outputs an element of $R(x)$ (rather than $\perp$). Specifically, the latter probability equals the aforementioned average number (which equals $|R(x)|/2^m$) divided by 1000ℓ. Using $m \leq \max(0, \log_2(2|R(x)|) - 6 - \log_2 \ell)$, we have $|R(x)|/2^m \geq \max(1, 32\ell)$, which means that the procedure outputs some element of $R(x)$ with probability at least $1/1000\ell$. ■

Technical comments. We can easily improve the performance of Construction 19 by dealing separately with the case $m = 0$. In such a case, Step 3 can be simplified and improved by uniformly selecting and outputting an element of S_λ (which equals $R(x)$). Recall that the probability that a uniform generation procedure outputs $\perp$ can be deceased by repeated invocations.

Appendix: On Hashing

Hashing is extensively used in complexity theory in order to map arbitrary (unstructured) sets "almost uniformly" to a smaller structured set of adequate size. Specifically, hashing is supposed to map an arbitrary 2^m-subset (of $\{0,1\}^n$) to $\{0,1\}^m$ in an "almost uniform" manner.

For a fixed set S of cardinality 2^m, a 1-1 mapping $f_S : S \to \{0,1\}^m$ does exist, but it is not necessarily an efficient one (e.g., it may require "knowing" the entire set S). Clearly, no fixed function $f : \{0,1\}^n \to \{0,1\}^m$ can map every 2^m subset of $\{0,1\}^n$ to $\{0,1\}^m$ in a 1-1 manner (or even approximately so). However, a random function $f : \{0,1\}^n \to \{0,1\}^m$ has the property that, for every 2^m-subset $S \subset \{0,1\}^n$, with overwhelmingly high probability f maps S to $\{0,1\}^m$ such that no point in the range has many f-preimages in S. The problem is that a truly random function is unlikely to have a succinct representation (let alone an efficient evaluation algorithm). We seek families of functions that have a similar property, but do have a succinct representation as well as an efficient evaluation algorithm.

A.1 Definitions

Motivated by the foregoing discussion, we consider families of functions $\{H_n^m\}_{m<n}$ Such that the following properties hold:

1. For every $S \subset \{0,1\}^n$, with high probability, a function h selected uniformly in H_n^m maps S to $\{0,1\}^m$ in an "almost uniform" manner. For example, for any $|S| = 2^m$ and each point y, with high probability over the choice of h, it holds that $|\{x \in S : h(x) = y\}| \leq \text{poly}(n)$.

2. The functions in H_n^m have succinct representation. For example, we may require that $H_n^m \equiv \{0,1\}^{\ell(n,m)}$, for some polynomial ℓ.

3. The functions in H_n^m can be efficiently evaluated. That is, there exists a polynomial-time algorithm that, on input a representation of a function, h (in H_n^m), and a string $x \in \{0,1\}^n$, returns $h(x)$. In some cases we make even more stringent requirements regarding the the algorithm (e.g., that it runs in linear space).

Condition 1 was left vague on purpose. At the very least, we require that the expected size of $\{x \in S : h(x) = y\}$ equals $|S|/2^m$. We shall see (in Section A.3) that different (stronger) interpretations of Condition 1 are satisfied by different types of hashing functions. We focus on t-wise independent hashing functions, defined next.

Definition 21 (t-wise independent hashing functions): *A family H_n^m of functions from n-bit strings to m-bit strings is called* t-wise independent *if for every t distinct domain elements $x_1, ..., x_t \in \{0,1\}^n$ and every $y_1, ..., y_t \in \{0,1\}^m$ it holds that*

$$\mathsf{Pr}_{h \in H_n^m}[\wedge_{i=1}^t h(x_i) = y_1] \;=\; 2^{-t \cdot m}$$

That is, every t domain elements are mapped by a uniformly chosen $h \in H_n^m$ in a totally uniform manner. Note that for $t \geq 2$, it follows that the probability that a random $h \in H_n^m$ maps two distinct domain elements to the same image is 2^{-m}. Such (families of) functions are called universal (cf. [3]), but we will focus on the stronger condition of t-wise independence.

A.2 Constructions

The following constructions are merely a re-interpretation of the constructions of pairwise-independent random variables. (Alternatively, one may view the latter constructions as a re-interpretation of the following two constructions.)

Construction 22 (t-wise independent hashing): *For $t, m, n \in \mathbb{N}$ such that $m \leq n$, consider the following family of hashing functions mapping n-bit strings to m-bit strings. Each t-sequence $\overline{s} = (s_0, s_1, ..., s_{t-1}) \in \{0,1\}^{t \cdot n}$ describes a function $h_{\overline{s}} : \{0,1\}^n \to \{0,1\}^m$ such that $h_{\overline{s}}(x)$ equals the m-bit prefix of the binary representation of $\sum_{j=0}^{t-1} s_j x^j$, where the arithmetic is that of* $\mathrm{GF}(2^n)$*, the finite field of 2^n elements.*

Construction 22 constitutes a family of t-wise independent hash functions. Typically, we will use either $t = 2$ or $t = \Theta(n)$. To make the construction totally explicit, we need an explicit representation of $\mathrm{GF}(2^n)$. An alternative construction for the case of $t = 2$ may be obtained analogously to a pairwise independent generator that is based on Toeplitz matrices. A Toeplitz matrix is a matrix with all diagonals being homogeneous; that is, $T = (t_{i,j})$ is a Toeplitz matrix if $t_{i,j} = t_{i+1,j+1}$, for all i, j.

Construction 23 (Alternative pairwise independent hashing): *For $m \leq n$, consider the family of hashing functions in which each n-by-m Toeplitz matrix T and an m-dimensional vector b describes a function $h_{T,b} : \{0,1\}^n \to \{0,1\}^m$ such that $h_{T,b}(x) = Tx + b$.*

Construction 23 constitutes a family of pairwise independent hash functions. Note that a n-by-m Toeplitz matrix can be specified by $n + m - 1$ bits, yielding description length $n + 2m - 1$. An alternative construction (using $m \cdot n + m$ bits of representation) uses arbitrary n-by-m matrices rather than Toeplitz matrices.

A.3 The Leftover Hash Lemma

We now turn to the "almost uniform" cover condition (i.e., Condition 1) mentioned in Section A.1. One concrete interpretation of this condition is implied by the following lemma.

Lemma 24 *Let $m < n$ be integers, H_n^m be a family of pairwise independent hash functions, and $S \subseteq \{0,1\}^n$. Then, for every $y \in \{0,1\}^m$ and every $\varepsilon > 0$, for all but at most an $\frac{2^m}{\varepsilon^2 |S|}$ fraction of $h \in H_n^m$ it holds that*

$$|\{x \in S : h(x) = y\}| = (1 \pm \varepsilon) \cdot \frac{|S|}{2^m} \tag{15}$$

By pairwise independence (or rather even by "1-wise independence"), the expected size of $\{x \in S : h(x) = y\}$ is $|S|/2^m$, where the expectation is taken uniformly over all $h \in H_n^m$. The lemma upper bounds the fraction of h's that deviate from the expected value. Needless to say, the bound is meaningful only in case $|S| > 2^m$ (or alternatively for $\varepsilon > 1$). Setting $\varepsilon = \sqrt[3]{2^m/|S|}$ (and focusing on the case that $|S| > 2^m$), we infer that *for all but at most an ε fraction of $h \in H_n^m$ it holds that* $|\{x \in S : h(x) = y\}| = (1 \pm \varepsilon) \cdot |S|/2^m$. Thus, each range element has approximately the right number of h-preimages in the set S under almost all $h \in H_n^m$.

Proof: Fixing an arbitrary set $S \subseteq \{0,1\}^n$ and an arbitrary $y \in \{0,1\}^m$, we estimate the probability that a uniformly selected $h \in H_n^m$ violates Eq. (15). We define random variables ζ_x, over the aforementioned probability space, such that $\zeta_x = \zeta_x(h)$ equal 1 if $h(x) = y$ and 0 otherwise. The expected value of $\sum_{x \in S} \zeta_x$ is $\mu \stackrel{\text{def}}{=} |S| \cdot 2^{-m}$, and we are interested in the probability that this sum deviates from the expectation. Applying Chebyshev's Inequality, we get

$$\mathsf{Pr}\left[\left|\mu - \sum_{x \in S} \zeta_x\right| > \varepsilon \cdot \mu\right] \;<\; \frac{\mu}{\varepsilon^2 \mu^2}$$

because $\mathsf{Var}(\sum_{x \in S} \zeta_x) < |S| \cdot 2^{-m}$ by the pairwise independence of the ζ_x's and the fact that $\mathsf{E}[\zeta_x] = 2^{-m}$. The lemma follows. ■

A generalization (called mixing). The proof of Lemma 24 can be easily extended to show that *for every set $T \subset \{0,1\}^m$ and every $\varepsilon > 0$, for all but at most an $\frac{2^m}{|T| \cdot |S| \varepsilon^2}$ fraction of $h \in H_n^m$ it holds that* $|\{x \in S : h(x) = y\}| = (1 \pm \varepsilon) \cdot |T| \cdot |S|/2^m$. (Hint: just define $\zeta_x = \zeta(h) = 1$ if $h(x) \in T$ and 0 otherwise.) In the case that $m = n$, this is called a mixing property, and is meaningfull provided $|T| \cdot |S| > 2^m/\varepsilon$.

An extremely useful corollary. The aforementioned generalization of Lemma 24 asserts that most functions behave well with respect to any fixed sets of preimages $S \subset \{0,1\}^n$ and images $T \subset \{0,1\}^m$. A seemingly stronger statement, which is (non-trivially) implied by Lemma 24, is that for all adequate sets S most functions $h \in H_n^m$ map S to $\{0,1\}^m$ in an almost uniform manner.[11] This is a consequence of the following theorem.

Theorem 25 (a.k.a Leftover Hash Lemma): *Let H_n^m and $S \subseteq \{0,1\}^n$ be as in Lemma 24, and define $\varepsilon = \sqrt[3]{2^m/|S|}$. Consider random variable X and H that are uniformly distributed on S and H_n^m, respectively. Then, the statistical distance between $(H, H(X))$ and (H, U_m) is at most 2ε.*

Using the terminology of randomness extractors, we say that H_n^m yields a strong extractor (with rather poor parameters).

Proof: Let V denote the set of pairs (h, y) that violate Eq. (15), and $\overline{V} \stackrel{\rm def}{=} (H_n^m \times \{0,1\}^m) \setminus V$. Then for every $(h, y) \in \overline{V}$ it holds that

$$\begin{aligned} \mathsf{Pr}[(H, H(X)) = (h, y)] &= \mathsf{Pr}[H = h] \cdot \mathsf{Pr}[h(X) = y] \\ &= (1 \pm \varepsilon) \cdot \mathsf{Pr}[(H, U_m) = (h, y)]. \end{aligned}$$

On the other hand, by Lemma 24 (which asserts $\mathsf{Pr}[(H, y) \in V] \leq \varepsilon$ for every $y \in \{0,1\}^m$), we have $\mathsf{Pr}[(H, U_m) \in V] \leq \varepsilon$. Using

$$\begin{aligned} \mathsf{Pr}[(H, H(X)) \in V] &= 1 - \mathsf{Pr}[(H, H(X)) \in \overline{V}] \\ &\leq 1 - \mathsf{Pr}[(H, U_n)) \in \overline{V}] + \varepsilon \;\leq\; 2\varepsilon \end{aligned}$$

we upper-bounded the statistical difference between $(H, H(X))$ and (H, U_m) by

$$\begin{aligned} &\frac{1}{2} \cdot \sum_{(h,y) \in H_n^m \times \{0,1\}^m} |\mathsf{Pr}[(H, H(X)) = (h, y)] - \mathsf{Pr}[(H, U_m) = (h, y)]| \\ &\leq \frac{\varepsilon}{2} + \frac{1}{2} \cdot \sum_{(h,y) \in V} |\mathsf{Pr}[(H, H(X)) = (h, y)] - \mathsf{Pr}[(H, U_m) = (h, y)]| \\ &\leq \frac{\varepsilon}{2} + \frac{1}{2} \cdot \sum_{(h,y) \in V} (\mathsf{Pr}[(H, H(X)) = (h, y)] + \mathsf{Pr}[(H, U_m) = (h, y)]) \\ &\leq \frac{\varepsilon}{2} + \frac{1}{2} \cdot (2\varepsilon + \varepsilon) \end{aligned}$$

and the claim follows. ■

An alternative proof of Theorem 25. Define the collision probability of a random variable Z, denote $\mathsf{cp}(Z)$, as the probability that two independent samples of Z yield the same result. Alternatively, $\mathsf{cp}(Z) \stackrel{\rm def}{=} \sum_z \mathsf{Pr}[Z = z]^2$. Theorem 25 follows by combining the following two facts:

1. A general fact: *If $Z \in [N]$ and $\mathsf{cp}(Z) \leq (1+4\epsilon^2)/N$ then Z is ϵ-close to the uniform distribution on $[N]$.*

   We prove the contra-positive: Assuming that the statistical distance between Z and the uniform distribution on $[N]$ equals δ, we show that $\mathsf{cp}(Z) \geq (1 + 4\delta^2)/N$. This is done by

[11] That is, for X as in Theorem 25 and any $\alpha > 0$, for all but at most an α fraction of the functions $h \in H_n^m$ it holds that $h(X)$ is $(2\varepsilon/\alpha)$-close to U_m.

defining $L \stackrel{\rm def}{=} \{z : \mathsf{Pr}[Z=z] < 1/N\}$, and lower-bounding $\mathtt{cp}(Z)$ by using the fact that the collision probability minimizes on uniform distributions. Specifically,

$$\mathtt{cp}(Z) \geq |L| \cdot \left(\frac{\mathsf{Pr}[Z \in L]}{|L|}\right)^2 + (N - |L|) \cdot \left(\frac{\mathsf{Pr}[Z \in [N] \setminus L]}{N - |L|}\right)^2,$$

which equals $1 + (\delta^2/(1-\rho)\rho) \geq 1 + 4\delta^2$, where $\rho = |L|/N$.

2. The collision probability of $(H, H(X))$ is at most $(1 + (2^m/|S|))/(|H_n^m| \cdot 2^m)$. (Furthermore, this holds even if H_n^m is only universal.)

   The proof is by a straightforward calculation. Specifically, note that $\mathtt{cp}(H, H(X)) = |H_n^m|^{-1} \cdot \mathsf{E}_{h \in H_n^m}[\mathtt{cp}(h(X))]$, whereas $\mathsf{E}_{h \in H_n^m}[\mathtt{cp}(h(X)] = |S|^{-2} \sum_{x_1,x_2 \in S} \mathsf{Pr}[H(x_1) = H(x_2)]$. The sum equals $|S| + (|S|^2 - |S|) \cdot 2^{-m}$, and so $\mathtt{cp}(H, H(X)) < |H_n^m|^{-1} \cdot (2^{-m} + |S|^{-1})$.

Note that it follows that $(H, H(X))$ is $\sqrt{2^m/4|S|}$-close to (H, U_m), which is a stronger bound than the one provided in Theorem 25.

Stronger uniformity via higher independence. Recall that Lemma 24 asserts that for each point in the range of the hash function, with high probability over the choice of the hash function, *this fixed point* has approximately the expected number of preimages in S. A stronger condition asserts that, with high probability over the choice of the hash function, *every point in its range* has approximately the expected number of preimages in S. Such a guarantee can be obtained when using n-wise independent hashing functions.

Lemma 26 *Let $m < n$ be integers, H_n^m be a family of n-wise independent hash functions, and $S \subseteq \{0,1\}^n$. Then, for every $\varepsilon \in (0,1)$, for all but at most an $2^m \cdot (n \cdot 2^m/\varepsilon^2|S|)^{n/2}$ fraction of $h \in H_n^m$, it holds that $|\{x \in S : h(x) = y\}| = (1 \pm \varepsilon) \cdot |S|/2^m$ for every $y \in \{0,1\}^m$.*

Indeed, the lemma should be used with $2^m < \varepsilon^2|S|/4n$. In particular, using $m = \log_2|S| - \log_2(5n/\varepsilon^2)$ guarantees that with high probability each range elements has $(1 \pm \varepsilon) \cdot |S|/2^m$ preimages in S. Under this setting of parameters $|S|/2^m = 5n/\varepsilon^2$, which is $\mathrm{poly}(n)$ whenever $\varepsilon = 1/\mathrm{poly}(n)$. Needless to say, this guarantee is stronger than the conclusion of Theorem 25.

Proof: The proof follows the footsteps of the proof of Lemma 24, taking advantage of the fact that the random variables (i.e., the ζ_x's) are now $2t$-wise independent, where $t = n/2$. This allows for the use of a so-called $2t^{\rm th}$ moment analysis, which generalizes the analysis of pairwise independent sampling. As in the proof of Lemma 24, we fix any S and y, and define $\zeta_x = \zeta_x(h) = 1$ if and only if $h(x) = y$. Letting $\mu = \mathsf{E}[\sum_{x \in S} \zeta_x] = |S|/2^m$ and $\overline{\zeta}_x = \zeta_x - \mathsf{E}(\zeta_x)$, we start with Markov inequality:

$$\begin{aligned} \mathsf{Pr}\left[\left|\mu - \sum_{x \in S} \zeta_x\right| > \varepsilon \cdot \mu\right] &< \frac{\mathsf{E}[(\sum_{x \in S} \overline{\zeta}_x)^{2t}]}{\varepsilon^{2t}\mu^{2t}} \\ &= \frac{\sum_{x_1,\ldots,x_{2t} \in S} \mathsf{E}[\prod_{i=1}^{2t} \overline{\zeta}_{x_i}]}{\varepsilon^{2t} \cdot (|S|/2^m)^{2t}} \qquad (16) \end{aligned}$$

Using $2t$-wise independence, we note that only the terms in Eq. (16) that do not vanish are those in which each variable appears with multiplicity. This mean that only terms having less than t distinct variables contribute to Eq. (16). Now, for every $j \leq t$, we have less than $\binom{|S|}{j} \cdot (2t!) < (2t!/j!) \cdot |S|^j$

terms with j distinct variables, and each contributes less than $(2^{-m})^j$ to the sum. Thus, Eq. (16) is upper-bounded by

$$\frac{2t!}{(\varepsilon^{2t}|S|/2^m)^{2t}} \cdot \sum_{j=1}^{t} \frac{(|S|/2^m)^j}{j!} \;<\; \frac{2t!/t!}{(\varepsilon^2|S|/2^m)^t} \;<\; \left(\frac{2t \cdot 2^m}{\varepsilon^2|S|}\right)^t$$

where the first inequality assumes $|S| > n2^m$ (since the claim hold vacuously otherwise). This upper-bounds the probability that a random $h \in H_n^m$ violates the mapping condition regarding a fixed y. Using a union bound on all $y \in \{0,1\}^m$, the lemma follows. ■

Notes

The counting class $\#\mathcal{P}$ was introduced by Valiant [15], who proved that computing the permanent of 0/1-matrices is $\#\mathcal{P}$-complete (i.e., Theorem 7). Interestingly, like in the case of Cook's introduction of NP-completeness [4], Valiant's motivation was determining the complexity of a specific problem (i.e., the permanent).

The proofs of Theorems 7 and 3 were omitted from the current text. Such proofs can be found in the origianl papers (i.e., [15] and [14], respectively), but we prefer our own presentation (given in [6]).

The approximation procedure for $\#\mathcal{P}$ is due to Stockmeyer [13], following an idea of Sipser [12]. Our exposition, however, follows further developments in the area. The randomized reduction of $\mathcal{NP}$ to problems of unique solutions was discovered by Valiant and Vazirani [16]. Again, our exposition is a bit different.

The connection between approximate counting and uniform generation (presented in Section 4.1) was discovered by Jerrum, Valiant, and Vazirani [9], and is applicable also beyond the setting of Theorem 18 (e.g., in the "Markov Chain approach" (see [11, Sec. 11.3.1])). The direct solution to uniform generation (presented in Section 4.2) is taken from [1].

In continuation to Section 2.1, which is based on [10], we refer the interested reader to [8], which presents a probabilistic polynomial-time algorithm for approximating the permanent of non-negative matrices. This fascinating algorithm is based on the fact that knowing (approximately) certain parameters of a non-negative matrix M allows to approximate the same parameters for a matrix M', provided that M and M' are sufficiently similar. Specifically, M and M' may differ only on a single entry, and the ratio of the corresponding values must be sufficiently close to one. Needless to say, the actual observation (is not generic but rather) refers to specific parameters of the matrix, which include its permanent. Thus, given a matrix M for which we need to approximate the permanent, we consider a sequence of matrices $M_0, ..., M_t \approx M$ such that M_0 is the all 1's matrix (for which it is easy to evaluate the said parameters), and each M_{i+1} is obtained from M_i by reducing some adequate entry by a factor sufficiently close to one. This process of (polynomially many) gradual changes, allows to transform the dummy matrix M_0 into a matrix M_t that is very close to M (and hence has a permanent that is very close to the permanent of M). Thus, approximately obtaining the parameters of M_t allows to approximate the permanent of M.

Exercises

Exercise 27 (enumeration problems) For any binary relation R, define the enumeration problem of R as a function $f_R : \{0,1\}^* \times \mathbb{N} \to \{0,1\}^* \cup \{\perp\}$ such that $f_R(x,i)$ equals the i^{th} element in $|R(x)|$ if $|R(x)| \geq i$ and $f_R(x,i) = \perp$ otherwise. The above definition refers to the standard lexicographic order on strings, but any other efficient order of strings will do.[12]

1. Prove that, for any polynomially bounded R, computing $\#R$ is reducible to computing f_R.

2. Prove that, for any $R \in \mathcal{PC}$, computing f_R is reducible to some problem in $\#\mathcal{P}$.

   Guideline: Consider the binary relation $R' = \{(\langle x,b\rangle, y) : (x,y) \in R \wedge y \leq b\}$, and show that f_R is reducible to $\#R'$. (Extra hint: Note that $f_R(x,i) = y$ if and only if $|R'(\langle x,y\rangle)| = i$ and for every $y' < y$ it holds that $|R'(\langle x,y'\rangle)| < i$.)

Exercise 28 (computing the permanent of integer matrices) Prove that computing the permanent of matrices with 0/1-entries is computationally equivalent to computing the number of perfect matchings in bipartite graphs.
(Hint: Given a bipartite graph $G = ((X,Y),E)$, consider the matrix M representing the edges between X and Y (i.e., the (i,j)-entry in M is 1 if the i^{th} vertex of X is connected to the j^{th} entry of Y), and note that only perfect matchings in G contribute to the permanent of M.)

Exercise 29 (computing the permanent modulo 3) Combining Proposition 8 and Theorem 16, prove that for every integer $n > 1$ that is relatively prime to c, computing the permanent modulo n is NP-hard under randomized reductions.[13] Since Proposition 8 holds for $c = 2^{10}$, hardness holds for every odd integer $n > 1$.

Guideline: Applying the reduction of Proposition 8 to the promise problem of deciding whether a 3CNF formula has a unique satisfiable assignment or is unsatisfiable. Use the fact that n does not divide any power of c.

Exercise 30 (negative values in Proposition 8) Assuming $\mathcal{P} \neq \mathcal{NP}$, prove that Proposition 8 cannot hold for a set I containing only non-negative integers. Note that the claim holds even if the set I is not finite (and even if I is the set of all non-negative integers).

Guideline: A reduction as in Proposition 8 provides a Karp-reduction of `3SAT` to deciding whether the permanent of a matrix with entries in I is non-zero. Note that the permanent of a *non-negative* matrix is non-zero if and only if the corresponding bipartite graph has a perfect matching.

Exercise 31 (error reduction for approximate counting) Show that the error probability δ in Definition 11 can be reduced from 1/3 (or even $(1/2) + (1/\text{poly}(|x|))$ to $\exp(-\text{poly}(|x|))$.

Guideline: Invoke the weaker procedure for an adequate number of times and take the *median* value among the values obtained in these invocations.

Exercise 32 (relative approximation for DNF satisfaction) Referring to the text of Section 2.1, prove the following claims.

1. Both assumptions regarding the general setting hold in case $S_i = C_i^{-1}(1)$, where $C_i^{-1}(1)$ denotes the set of truth assignments that satisfy the conjunction C_i.

   Guideline: In establishing the second assumption note that it reduces to the conjunction of the following two assumptions:

[12] An order of strings is a 1-1 and onto mapping μ from the natural numbers to the set of all strings. Such order is called efficient if both μ and its inverse are efficiently computable. The standard lexicographic order satisfies $\mu(i) = y$ if the (compact) binary expansion of i equals $1y$; that is $\mu(1) = \lambda$, $\mu(2) = 0$, $\mu(3) = 1$, $\mu(4) = 00$, etc.

[13] Actually, a sufficient condition is that n does not divide any power of c. Thus (referring to $c = 2^{10}$), hardness holds for every integer $n > 1$ that is not a power of 2. On the other hand, for any fixed $n = 2^e$, the permanent modulo n can be computed in polynomial-time [15, Thm. 3].

(a) Given i, one can efficiently generate a uniformly distributed element of S_i. Actually, generating a distribution that is almost uniform over S_i suffices.

(b) Given i and x, one can efficiently determine whether $x \in S_i$.

2. Prove Proposition 13, relating to details such as the error probability in an implementation of Construction 12.

3. Note that Construction 12 does not require exact computation of $|S_i|$. Analyze the output distribution in the case that we can only approximate $|S_i|$ up-to a factor of $1 \pm \varepsilon'$.

Exercise 33 (reducing the relative deviation in approximate counting) Prove that, for any $R \in \mathcal{PC}$ and every polynomial p and constant $\delta < 0.5$, there exists $R' \in \mathcal{PC}$ such that $(1/p, \delta)$-approximation for $\#R$ is reducible to $(1/2, \delta)$-approximation for $\#R'$.

Guideline: For $t(n) = \Omega(p(n))$, let $R' = \{(x, (y_1, ..., y_{t(|x|)})) : (\forall i)\,(x, y_i) \in R\}$. Note that $|R(x)| = |R'(x)|^{1/t(|x|)}$, and thus if $a = (1 \pm (1/2)) \cdot |R'(x)|$ then $a^{1/t(|x|)} = (1 \pm (1/2))^{1/t(|x|)} \cdot |R(x)|$. Furthermore, prove that $(1/p, \delta)$-approximation for $\#R$ is reducible to approximating $\#R''$ to within a factor of $F(n) = \exp(p(n))$ with error probability δ, for some $\mathbb{R}'' \in \mathcal{PC}$.
(Hint: Same as the main part. Note that the length of the solution for $R''(x)$ is larger than $p(|x|)$ and so there is nothing wrong in approximating $\#R''(|x|)$ to within $F(|x|)$.)

Exercise 34 (deviation reduction in approximate counting, cont.) In continuation to Exercise 33, prove that if R is NP-complete via parsimonious reductions then, for every positive polynomial p and constant $\delta < 0.5$, the problem of $(1/p, \delta)$-approximation for $\#R$ is reducible to $(1/2, \delta)$-approximation for $\#R$.
(Hint: Compose the reduction (to the problem of $(1/2, \delta)$-approximation for $\#R'$) provided in Exercise 33 with the parsimonious reduction of $\#R'$ to $\#R$.)
Prove that, for every function F' such that $F'(n) = \exp(n^{o(1)})$, we can also reduce the aforementioned problems to the problem of approximating $\#R$ to within a factor of F' with error probability δ.

Guideline: Using R'' as in Exercise 33, we encounter a technical difficulty. The issue is that the composition of the ("amplifying") reduction of $\#R$ to $\#R''$ with the parsimonious reduction of $\#R''$ to $\#R$ may increase the length of the instance. Indeed, the length of the new instance is polynomial in the length of the original instance, but this polynomial may depend on R'', which in turn depends on F'. Thus, we cannot use $F'(n) = \exp(n^{1/O(1)})$ but $F'(n) = \exp(n^{o(1)})$ is fine.

Exercise 35 Referring to the procedure in the proof Theorem 14, show how to use an NP-oracle in order to determine whether the number of solutions that "pass a random sieve" is greater than t. You are allowed queries of length polynomial in the length of x, h and in the size of t.
(Hint: Consider the set $S'_{R,H} \stackrel{\text{def}}{=} \{(x, i, h, 1^t) : \exists y_1, ..., y_t \text{ s.t. } \psi'(x, h, y_1, ..., y_t)\}$, where $\psi'(x, h, y_1, ..., y_t)$ holds if and only if the y_j are different and for every j it holds that $(x, y_j) \in R \wedge h(y_j) = 0^i$.)

Exercise 36 (parsimonious reductions and Theorem 16) Demonstrate the importance of parsimonious reductions in Theorem 16 by proving the following:

1. There exists a search problem $R \in \mathcal{PC}$ such that every problem in $\mathcal{PC}$ is reducible to R (by a non-parsimonious reduction) and still the the promise problem $(\mathtt{US}_R, \overline{S}_R)$ is decidable in polynomial-time.

   Guideline: Consider the following artificial witness relation R for $\mathtt{SAT}$ in which $(\phi, \sigma\tau) \in R$ if $\sigma \in \{0, 1\}$ and τ satisfies ϕ. Note that the standard witness relation of $\mathtt{SAT}$ is reducible to R, but this reduction is not parsimonious. Also note that $\mathtt{US}_R = \emptyset$ and thus $(\mathtt{US}_R, \overline{S}_R)$ is trivial.

2. There exists a search problem $R \in \mathcal{PC}$ such that $\#R$ is $\#\mathcal{P}$-complete and still the the promise problem $(\mathtt{US}_R, \overline{S}_R)$ is decidable in polynomial-time.

   Guideline: One easy proof is to use the relation suggested in the guideline to Part 1. A totally different proof relies on Theorem 7 and on the fact that it is easy to decide $(\mathtt{US}_R, \overline{S}_R)$ when R is the corresponding perfect matching relation (by computing the determinant).

Exercise 37 Prove that SAT is randomly reducible to deciding unique solution for SAT, *without using the fact that* SAT *is NP-complete via parsimonious reductions.*

Guideline: Follow the proof of Theorem 16, while using the family of pairwise independent hashing functions provided in Construction 23. Note that, in this case, the condition $(\tau \in R_{\mathtt{SAT}}(\phi)) \wedge (h(\tau)=0^i)$ can be directly encoded as a CNF formula. That is, consider the formula ϕ_h such that $\phi_h(z) \stackrel{\rm def}{=} \phi(z) \wedge (h(z)=0^i)$, and note that $h(z)=0^i$ can be written as the conjunction of i clauses, where each clause is a CNF that is logically equivalent to the parity of some of the bits of z (where the identity of these bits is determined by h).

Exercise 38 (an alternative procedure for approximate counting) Adapt Step 1 of Construction 19 so to obtain an approximate counting procedure for $\#R$.

Guideline: For $m = 0, 1, ...\ell$, the procedure invokes Step 1 of Construction 19 until a negative answer is obtained, and outputs 2^m for the current value of m. For $|R(x)| > 1000\ell$, this yields a constant factor approximation of $|R(x)|$. In fact, we can obtain a better estimate by making additional queries at iteration m (i.e., queries of the form $(x, h, 1^i)$ for $i = 10\ell, ..., 1000\ell$). The case $|R(x)| \leq 1000\ell$ can be treated by using Step 2 of Construction 19, in which case we obtain an exact count.

References

[1] M. Bellare, O. Goldreich, and E. Petrank. Uniform Generation of NP-witnesses using an NP-oracle. *Information and Computation*, Vol. 163, pages 510–526, 2000.

[2] S. Ben-David, B. Chor, O. Goldreich, and M. Luby. On the Theory of Average Case Complexity. *Journal of Computer and System Science*, Vol. 44 (2), pages 193–219, 1992.

[3] L. Carter and M. Wegman. Universal Hash Functions. *Journal of Computer and System Science*, Vol. 18, 1979, pages 143–154.

[4] S.A. Cook. The Complexity of Theorem Proving Procedures. In *3rd ACM Symposium on the Theory of Computing*, pages 151–158, 1971.

[5] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

[6] O. Goldreich. Expositions in Complexity Theory (various texts). Unpublished notes, December 2005. Availabe from the webpage `http://www.wisdom.weizmann.ac.il/~oded/cc-texts.html`

[7] O. Goldreich, S. Vadhan and A. Wigderson. On interactive proofs with a laconic provers. *Computational Complexity*, Vol. 11, pages 1–53, 2002.

[8] M. Jerrum, A. Sinclair, and E. Vigoda. A Polynomial-Time Approximation Algorithm for the Permanent of a Matrix with Non-Negative Entries. *Journal of the ACM*, Vol. 51 (4), pages 671–697, 2004.

[9] M. Jerrum, L. Valiant, and V.V. Vazirani. Random Generation of Combinatorial Structures from a Uniform Distribution. *Theoretical Computer Science*, Vol. 43, pages 169–188, 1986.

[10] R.M. Karp and M. Luby. Monte-Carlo algorithms for enumeration and reliability problems. In *24th IEEE Symposium on Foundations of Computer Science*, pages 56-64, 1983.

[11] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[12] M. Sipser. A Complexity Theoretic Approach to Randomness. In *15th ACM Symposium on the Theory of Computing*, pages 330–335, 1983.

[13] L. Stockmeyer. The Complexity of Approximate Counting. In *15th ACM Symposium on the Theory of Computing*, pages 118–126, 1983.

[14] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, Vol. 20 (5), pages 865–877, 1991.

[15] L.G. Valiant. The Complexity of Computing the Permanent. *Theoretical Computer Science*, Vol. 8, pages 189–201, 1979.

[16] L.G. Valiant and V.V. Vazirani. NP Is as Easy as Detecting Unique Solutions. *Theoretical Computer Science*, Vol. 47 (1), pages 85–93, 1986.