On the complexity of enumerating ordered sets

Oded Goldreich
Department of Computer Science
Weizmann Institute of Science
Rehovot, Israel.
oded.goldreich@weizmann.ac.il

September 14, 2023

Abstract

We consider the complexity of enumerating ordered sets, defined as solving the following type of a computational problem: For a predetermined ordered set, given $i \in \mathbb{N}$, one is required to answer with the i^{th} member of the set (according to the predetermined order).

Our focus is on countable sets such as the primes and the rational numbers, although in these cases we provide no decisive answers. In general, we do not report of any exciting results, but rather make a few observations and suggest some open problems.

Contents

1 Introduction and Overview

Most studies of the "complexity of enumeration" refer to enumerating the solutions to search problems (see, e.g., [8, 4]). Specifically, for a fixed relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$, given an input x, the task is to find *all* solutions to x (i.e., all elements of the set $R(x) \stackrel{\text{def}}{=} \{y : (x,y) \in R\}$), and the main complexity measure considered is the "delay" (i.e., time spent between outputting two consecutive solutions).[1] In particular, typically, the order in which these elements are generated is at the discretion of the algorithm's designer.[2]

The last point holds also with respect to previous studies of the problem of enumerating (or generating) the elements of some combinatorial set (see, e.g., [10, Sec. 7.2]).[3] For example, on input $n \in \mathbb{N}$, one may be asked to generate *all* permutations over $[n] \stackrel{\text{def}}{=} \{1, 2, ..., n\}$, whereas the order in which these permutations are generated is typically left to the algorithm's designer. In fact, the various algorithms suggested in [10, Sec. 7.2.1.2] are aimed at optimizing various "complexity" criteria.

In contrast, by *enumerating ordered sets* we mean enumerating the elements in a single *ordered* (countable) set *according to the predetermined* (total) *order*. Furthermore, we are interested in the *generation of individual elements* according to their location in the predetermined order. Specifically, for a fixed set $S \subseteq \{0,1\}^*$ and a fixed order on strings (e.g., the lexicographical order), we consider the task in which, given $i \in \mathbb{N}$, one is required to output the i^{th} element in S, denoted $\mathtt{enm}_S(i)$. We consider two notions of complexity, but our focus is on the first (i.e., strict) one.

***A strict notion*:** Here, the complexity of enumerating S is defined as the complexity of computing the function $\mathtt{enm}_S : \mathbb{N} \to \{0,1\}^*$, where (as usual) elements of $\mathbb{N}$ are presented in binary notation.

Typically (but not always), this notion is closely related to the complexity of counting the number of elements in S that precede a given string (according to the predetermied order): See Theorem 1.5.

***An amortized notion*:** Here, we actually consider the time complexity of producing the list of the first i elements (i.e., affecting the mapping $i \mapsto (\mathtt{enm}_S(1), ..., \mathtt{enm}_S(i))$), for every $i \in \mathbb{N}$. Dividing the total time by i yield a notion of *amortized complexity*.

We stress that information generated while producing the first $i-1$ elements can be used when producing the i^{th} element. A more stringent notion may refer to the *average time* it takes to evaluate $\mathtt{enm}_S$ on the first i numbers.

Recall that the prior works (e.g., [4, 8, 10]) are focused on computing the successive function (according to an order that is chosen by the algorithm's designer). We stress that the complexity of computing the successive function does not yield good upper bounds on the (strict) complexity of computing the corresponding enumeration function.

Our main point is that, while in some cases the order in which elements are enumerated is immaterial, there are cases in which a natural (predetermined) order matters. This is most evident

[1] In addition, one also considers the time to generate the first solution and the time to determine that no additional solution exists. Conceptually, we view these as special cases of the main complexity measure (i.e., "delay").

[2] We mention that both [8] and [10] refer to cases in which enumeration is according to a natural order (e.g., the lexicographic one), but they seem to view it as an extra feature.

[3] Given the vast volume of [10, Sec. 7.2], which spans more than 230 pages, it is possible that facts such as Observation 1.3 appear there.

in the case of enumerating "naturally" ordered sets such as the prime numbers and the rational numbers. More generally, in many cases, it is natural to require that elements having shorter description be enumerated before elements having longer description.

Another important point is that we focus on a single countable ordered set rather than on a countable sequence of finite sets. The reason that this technical difference matters is that the enumeration task is stated in terms of a single order, which is an order on the elements not on the sets. Specifically, given $i \in \mathbb{N}$, one is asked to return the i^{th} element in the ordered set (or the i first elements in the case of amortize complexity). In contrast, when considering a countable sequence of finite sets, one is asked to produce all elements in one of these sets, and is not required to figure out (implicitly) the number of elements in all prior sets.

1.1 The strict notion

Let us first spell out the weak relation between our focus and the focus of most prior studies.

Remark 1.1 (relation to enumerating solutions): *For a search problem $R \subseteq \{0,1\}^* \times \{0,1\}^*$, we consider the set $S = \{x0^{\ell(|x|)-|y|}1y : (x,y) \in R\}$, where $\ell : \mathbb{N} \to \mathbb{N}$ is a monotonically increasing function such that $\ell(n)$ is an upper bound on the length of solutions to n-bit long instances. Then, the complexity of enumerating solutions to R* (in the standard sense (i.e., as in [8, 4])) *is upper-bounded by the complexity of enumerating S* (according to the lexicographic order)*. Furthermore, letting $R(x) = \{y : (x,y) \in R\}$, we can generate the j^{th} string in $R(x)$ by first determining* (see Theorem 1.5) *the number of elements in S that precede $x0^{\ell(|x|)}1$, denoted n_x, and then finding the $n_x + j^{\text{th}}$ element in S. Note, however, that the converse does not hold.*

- *It may be harder to enumerate the solutions according to a predetermined order rather than arbitrarily* (i.e., according to the order determined by the enumerator itself)*.*
- *It may be possible to enumerate solutions to R* (i.e., enumerate $R(x)$ when given x) *without determining the number of solutions to all prior instances* (i.e., $\sum_{z<x} |R(z)|$)*.*

Note that enumerating (or generating) *the elements of some combinatorial set* (e.g., [10, Sec. 7.2]) *can be cast as a search problem.*[4]

Using the fact that efficiently enumerating an ordered set implies efficiently counting the number of elements in the set that precede a given string (see Theorem 1.5), it follows that *some enumeration problems are $\#\mathcal{P}$-hard*: This holds for the enumeration problem of a set S that is defined as in Remark 1.1 based on any search problem R for which counting the number of solutions is $\#\mathcal{P}$-hard.

Some examples. In contrast to Remark 1.1, we focus on the enumeration of "natural" countable sets, of the type that appear in standard Math textbooks. In order to emphasize this focus as well as offer a complexity-theoretic perspective on traditional claims of countability, we spell out a few simple example, listing them according to the complexity of enumeration.

1. Enumerating $\mathbb{N}$ itself. This task is, of course, trivial, since $\mathtt{enm}_{\mathbb{N}}(i) = i$.

[4] Typically, on input n, one is required to generate all elements of a finite set S_n. This can be captured by the search problem R such that $(x,y) \in R$ if and only if $y \in S_{|x|}$ (or, alternatively, by the search problem $\{(1^n, y) : n \in N \wedge y \in S_n\}$).

2. Enumerating the even natural numbers. This task is almost trivial (i.e., it is in $\mathcal{NC}^0$), since $\mathtt{enm}_{2\mathbb{N}}(i) = 2i$, which means that the algorithm amounts to appending a zero.

3. Enumerating the odd natural numbers. Here, given i we need to compute $2i-1$, which is slightly less trivial than computing $2i$ (e.g., it is in $\mathcal{AC}^0$ but not in $\mathcal{NC}^0$).

4. Enumerating the set $3\mathbb{N} = \{3n : n \in \mathbb{N}\}$. Here, given i we need to compute $3i$, which is slightly less trivial than computing $2i-1$. (E.g., computing $\mathtt{enm}_{3\mathbb{N}}$ requires width three ROBP, whereas $\mathtt{enm}_{2\mathbb{N}-1}$ can be computed by width two ROBP.)

5. Enumerating all (natural numbers that are) squares. Here, given i we need to compute i^2. This is clearly feasible; it is in $\mathcal{NC}_1$ (but not in $\mathcal{AC}^0[p]$ for any fixed prime p).[5]

6. Enumerating all (natural number that are) powers of two. Here, given i we need to compute 2^i, which means that the output length is exponential in the input length. This is clearly infeasible (i.e., cannot be done in polynomial time).

Examples 1–5 illustrate a fine complexity hierarchy among enumeration problems. In contrast to them, Example 6 illustrates an enumeration problem that requires exponential time, but this lower bound is due to the extreme sparsity of the target set (to be enumerated): the i^{th} element in it has length $i + 1 > 2^{|i|-1}$. A much more natural enumeration problem that seems very hard, although it is dense, is presented next.

Observation 1.2 (enumerating the prime numbers): *Enumerating the set of all prime numbers, according to their size, seems infeasible, because this problem is computationally equivalent to counting the number of primes that are smaller than a given natural number* (see Theorem 1.5). *The best known algorithm for the latter problem runs in exponential-time* (see [7] and the references therein). *We stress that the i^{th} prime number is of length $|i| + o(|i|)$, so* (unlike in Example 6) *the perceived difficulty of the enumeration task is not due to the length of the desired output.*

Before proceeding let us comment on the choice of the predetermined order. In general, we think of this order as being determined by the 'user' (i.e., the application) rather than by the algorithm's designer. Still, in some cases, standard (and natural) orders are extremely appealing.

Some standard orders. In the case of the natural numbers, the standard order is indeed the one according to which the natural numbers are actually defined (i.e., by use of the successive function). In the case of the set of all strings, the standard order is the lexicographical one (in which longer strings appear after shorter ones, and the bit 1 appears after the bit 0). Needless to say, these two standard orders (almost) coincide under the standard representation of natural numbers by binary strings.[6] When moving to t-tuples of natural numbers, a standard choice is that $(m_1, ..., m_t)$ precedes $(n_1, ..., n_t)$ if either $\sum_{i\in[t]} m_i < \sum_{i\in[t]} n_i$ or $\sum_{i\in[t]} m_i = \sum_{i\in[t]} n_i$ and for some $t' \in [t-1]$ it holds that $m_{t'} < n_{t'}$ and $m_i = n_i$ for all $i \in [t'-1]$. This order is used in standard texts that seek enumeration of $\mathbb{N}^t$, and we call it the standard order of t-tuples.

[5] Note that integer multiplication is $\mathcal{AC}^0$-reducible to squaring, whereas integer multiplication is not in $\mathcal{AC}^0[p]$. The latter fact holds because, for any prime $q \neq p$, *addition mod q*, which is hard for $AC^0[p]$, is $\mathcal{AC}^0$-reducible to integer multiplication.

[6] Actually, the most straightforward way of representing the natural numbers by (non-empty) strings avoids the 1-bit long string 0. Alternatively, $\mathbb{N} \cup \{0\}$ corresponds to $\{0,1\}^+$. Yet another correspondence is of $\mathbb{N}$ with $\{0,1\}^*$ (when dropping the leading 1 in the binary representation of numbers).

Observation 1.3 (enumerating tuples of natural numbers): *For any $t \in \mathbb{N}$, the set of t-tuples ordered according to the foregoing standard order can be enumerated in polynomial time. The key observation is that for every $s \in \mathbb{N}$ and $t' \in \mathbb{N}$, the cardinality of the set*

$$N_{s,t'} \stackrel{\text{def}}{=} \left\{ (q_1, ..., q_{t'}) \in \mathbb{N}^{t'} : \sum_{i \in [t']} q_i = s \right\} \tag{1}$$

is $N_{s,t'} \stackrel{\text{def}}{=} \binom{s-1}{t'-1}$, which is a polynomial of degree $t'-1$ in s with coefficients that only depend on t'. Hence, given $\overline{a} = (a_1, ..., a_t)$, we can compute the number of t-tuples preceding $\overline{a}$ in polynomial time, since this number can be represented by a polynomial of degree t in the a_i's with coefficients that only depend on t (or rather as a sum of t bivariate polynomials of degree t in some partial sums of the a_i's (with coefficients that only depend on t)).[7]

We are often interested in enumerating an easily recognizable subset of the set of t-tuples of natural numbers (e.g., ordered pairs representing the positive rational numbers). In light of Remark 1.1 (as well as Observation 1.2), this may not be feasible, even in the case of $t = 1$ (when insisting on the standard order).

On enumerating the rational numbers. Indeed, how about enumerating the positive rational numbers? Specifically, we associate the rational numbers with ordered pairs of natural numbers that are relatively prime, and seek to enumerate the set of these pairs according to the standard order of pairs (used in Observation 1.3).

An efficient algorithm for generating all rational numbers was presented in [2], but it generates these numbers in an order that is fundamentally different from the foregoing standard order. Specifically, the i^{th} rational number in the order defined by this algorithm is $f(i)/f(i+1)$ such that $f(1) = f(2) = 1$ whereas $f(2j+1) = f(j)$ and $f(2j+2) = f(j) + f(j+1)$. Observing that, for every $i \in \mathbb{N}$, the values $f(i+1)$ and $f(i)$ are easily computed from $f(\lceil i/2 \rceil)$ and $f(\lceil i/2 \rceil - 1)$, the efficiency of enumeration follows. For example, the ten first rational numbers in this order are

$$\frac{1}{1} \quad \frac{1}{2} \quad \frac{2}{1} \quad \frac{1}{3} \quad \frac{3}{2} \quad \frac{3}{1} \quad \frac{1}{4} \quad \frac{4}{3} \quad \frac{3}{5} \quad \frac{5}{2}$$

Furthermore, observing that $f(2^k - 1) = 1$ and $f(2^k) = k$, it follows that the 2^kth rational number in this order is $1/k$, which means that for $i = 2^k$ the length of the i^{th} pair is logarithmic in the length of the index (i.e., $|k| = \log_2 |i|$).

In contrast, we wish to enumerate pairs $(a, b) \in \mathbb{N}^2$ such that $\gcd(a, b) = 1$ but do so such that (a, b) appears before (a', b') if $a + b < a' + b'$. (Note that the i^{th} rational number in this order has description length $\Theta(|i|)$.) A key observation is that, for every integer $s \geq 2$, the set $\{(a, b) \in \mathbb{N}^2 : a + b = s \wedge \gcd(a, b) = 1\}$ is in 1-1 correspondence with the set $\{a \in [s-1] : \gcd(a, s) = 1\}$; hence, both sets have size $\varphi(s)$, where φ is Euler's function. As observed next, this suggests that it is infeasible to enumerate the rational numbers according to the aforementioned standard order.

[7] Specifically, for every $t' \in [t-1]$, the number of t-tuples of the form $(a_1, .., a_{t'-1}, b, b_{t'+1}, ..., b_t)$ such that $b + \sum_{i \in [t'+1,t]} b_i \leq a_{t'} + \sum_{i \in [t'+1,t]} a_i$ and $b < a_{t'}$ equals $\sum_{b \in [a_{t'}-1]} \sum_{s' \in [a'-b]} N_{s',t'}$, where $a' = \sum_{i \in [t'+1,t]} a_i$. Note that $\sum_{s' \in [x]} N_{s',t'}$ is a polynomial of degree t' in x with coefficients that only depend on t', whereas $\sum_{r \in [y]} \sum_{s' \in [x-r]} N_{s',t'}$ is a polynomial of total degree $t'+1$ in x and y (with coefficients that only depend on t').

Observation 1.4 (enumerating the rational numbers and computing Euler's function): *Consider the following three computational problems:*

1. *Enumerating the set* $\mathbb{Q} \stackrel{\text{def}}{=} \{(a,b) \in \mathbb{N}^2 : \gcd(a,b) = 1\}$ *according to the standard order on pairs of natural numbers; that is, computing* $\mathtt{enm}_{\mathbb{Q}}$.

2. *Computing the function* $s \mapsto \sum_{i \in [s]} \varphi(i)$, *where* φ *is Euler's function.*

3. *Computing Euler's function* (i.e., φ), *which is equivalent to factoring [11, Sec. 10.4].*[8]

Then, (3) *is reducible to* (2), *and* (2) *is reducible to* (1), *where the latter reduction relies on Theorem 1.5 and the fact that* $\sum_{i \in [s]} \varphi(i)$ *equals the number of pairs preceding* $(s, 1)$.

It is not clear whether (1) is reducible to (2), and whether (2) is reducible to (3).

On enumerating the integers. In contrast, it is easy to enumerate the *integers*, when represented as ordered pairs $(a,b) \in \mathbb{N}^2$ such that $\{a,b\} \ni 1$ and (a,b) represents $a - b$, according to the standard order on pairs. The first integer in this order is 0 (represented by $(1,1)$), whereas the $2j^{\text{th}}$ and $(2j+1)^{\text{st}}$ integers are j and $-j$ (represented by $(j+1,1)$ and $(1,j+1)$, respectively).

Enumerating versus counting. As mentioned already (see, e.g., Observation 1.2), the complexity of enumerating a set is closely related to the complexity of counting the number of members of the set that precede a given string.

Theorem 1.5 (enumerating versus counting, a special case): *For any polynomial-time recognizable set* S *such that the number of* n*-bit strings in* S *is at least* $\exp(n^{\Omega(1)})$, *the following two tasks are polynomial-time reducible to one another.*

1. *On input* $i \in \mathbb{N}$, *output the* i^{th} *string in* S.

2. *On input* $x \in \{0,1\}^*$, *output the number of strings in* S *that precede* x.

In both cases, the order is the lexicographic one.

In particular, Theorem 1.5 applies to the enumeration of the primes and the rational numbers (discussed in Observations 1.2 and 1.4, resp.). We mention that the density and efficient recognizability conditions are used only in proving that counting implies enumerating. While Theorem 1.5 refers to the lexicographic order only, more general results that refer to other orders are stated in Theorems 2.3 and 2.4.

[8] It is easy to see that factoring composites that are the product of two different primes reduces to evaluating φ on them: Letting $N = PQ$, for primes $P \neq Q$, it holds that $\varphi(N) = (P-1) \cdot (Q-1)$, which implies that $P + Q = N + 1 - \varphi(N)$. Hence, factoring N reduces to solving the quadratic equation $X \cdot (N + 1 - \varphi(N) - X) = N$.

1.2 The amortized notion

Note that enumerating S in amortized polynomial-time means that the first i elements are produced in time $i \cdot \mathrm{poly}(|i|) = \widetilde{O}(i)$. A sufficient condition for this to happen is that the set is easily recognizable and dense.

Theorem 1.6 (a sufficient condition): *Suppose that the set S is polynomial-time recognizable and $|S \cap \{0,1\}^n| \geq 2^n/\mathrm{poly}(n)$. Then, the amortized complexity of enumerating S* (according to the lexicographic order) *is polynomial.*

Note that low amortized complexity of enumerating S implies (very) *weak* forms of both conditions. In particular, it requires that among the i first elements in S almost all have at most $\mathrm{poly}(|i|)$ length, which implies $|S \cap \{0,1\}^\ell| \geq 2^{\ell^{\Omega(1)}}$. It also implies that S can be decided in exponential time, because $x \in S$ implies that x appears among the first $2^{|x|+1}$ elements (which means that it will be produced by the enumeration within $\widetilde{O}(2^{|x|+1})$ time).

Corollaries. Using Theorem 1.6, we observe that the amortized complexity of enumerating the following two ordered sets is polynomial.

1. The set of all primes (ordered according to their size).
2. The set of all positive rational numbers (represented by ordered pairs of natural numbers that are ordered as in Observations 1.3 and 1.4).

Needless to say, this stands in contrast to the perceived difficulty of enumerating these sets in the strict sense (cf. Observation 1.2 and 1.4).

1.3 Organization

Sections 2 and 3 provide more details about the notions discussed in the introduction. Specifically, Section 2 expands on the treatment of the strict notion that was discussed in Section 1.1, whereas Section 3 expands the discussion of the amortized notion provided in Section 1.2.

Section 4 takes a tangential issue that arises naturally when discussing the amortized notion, and Section 5 provides a compilation of some open problems.

2 The Strict Notion: More Details

In this section, we formulate more rigorously the notions that are mentioned in Section 1.1, generalize Theorem 1.5, and present a few additional efficient enumerations.

We start by clarifying that by order on strings we mean a correspondence between the set of strings and the set of natural numbers. Hence, an arbitrary order on strings is defined by an arbitrary bijection, called a *successor* function, from $\{0,1\}^*$ to $\{0,1\}^* \setminus \{\alpha\}$, where α is an arbitrary string, which is the first according to this order.

Definition 2.1 (the complexity of enumeration): *For an infinite countable set $S \subseteq \{0,1\}^*$ and an order on strings, denoted $\prec$, let $\mathtt{enm}^{\prec}_S(i)$ denote the i^{th} element in S* (i.e., $\mathtt{enm}^{\prec}_S(i) \in S$ and $|\{x \in S : x \prec \mathtt{enm}^{\prec}_S(i)\}| = i-1$). *Then, the* complexity of enumerating *S is defined as the complexity of evaluating the function $\mathtt{enm}^{\prec}_S$.*

For a set S and an order as in Definition 2.1, we define the ("counting") function $\mathtt{cnt}^{\prec}_S:\{0,1\}^*\to\mathbb{N}$ such that $\mathtt{cnt}^{\prec}_S(x) \stackrel{\rm def}{=} |\{z\in S : z\prec x\}|$. The following observations can be readily verified.

- For every $i\in\mathbb{N}$, it holds that $\mathtt{cnt}^{\prec}_S(\mathtt{enm}^{\prec}_S(i)) = i-1$.
- For every $x\in S$, if $x\in S$ then $\mathtt{enm}^{\prec}_S(\mathtt{cnt}^{\prec}_S(x)+1) = x$, and otherwise $x\prec\mathtt{enm}^{\prec}_S(\mathtt{cnt}^{\prec}_S(x)+1)$.

Let $\mathtt{succ}^{\prec}:\{0,1\}^*\to\{0,1\}^*$ denote the successor function; that is, $\mathtt{succ}^{\prec}(x)=y$ if $x\prec y$ and $x\prec z$ implies $y\preceq z$. Then:

- For every x, it holds that $x\in S$ if and only if $\mathtt{cnt}^{\prec}_S(x) < \mathtt{cnt}^{\prec}_S(\mathtt{succ}^{\prec}(x))$.

We observe that, typically, the counting function (i.e., $\mathtt{cnt}^{\prec}_S$) is in $\#\mathcal{P}$. Specifically, this holds for the lexicographic order and any set $S\in\mathcal{P}$.

Theorem 2.2 (typically, $S\in\mathcal{P}$ implies $\mathtt{cnt}^{\prec}_S\in\#\mathcal{P}$): *Let $\prec$ be a polynomial-time decidable order on strings* (i.e., given $x,y\in\{0,1\}^*$, we can decide in polynomial-time whether $x\prec y$ or $y\prec x$) *such that $y\prec x$ implies $|y|\leq\mathrm{poly}(|x|)$. Then, for every polynomial-time recognizable set S, it holds that $\mathtt{cnt}^{\prec}_S$ is in $\#\mathcal{P}$.*

Proof: Consider the search problem

$$R \stackrel{\rm def}{=} \{(x,y) : y\in S \wedge y\prec x\}.$$

Then, R is polynomially bounded (i.e., $(x,y)\in R$ implies $|y|\leq\mathrm{poly}(|x|)$ and polynomial-time recognizable, which means that $\#R$ is in $\#\mathcal{P}$. On the other hand, $\mathtt{cnt}^{\prec}_S(x) = |\{y\in S : y\prec x\}|$ equals the number of solutions for x (w.r.t R). ■

2.1 Generalization of Theorem 1.5

The two directions of Theorem 1.5 are captured by the following two theorems, which actually generalize it. These generalizations of Theorem 1.5 impose minimal restrictions on the order $\prec$ (hinted at by the word "typically" in the titles of Theorems 2.3 and 2.4). Needless to say, these restrictions are definitely satisfied by the lexicographic order and natural variations of it.

Theorem 2.3 (typically, enumerating S implies computing $\mathtt{cnt}^{\prec}_S$): *Let $\prec$ be a polynomial-time decidable order on strings* (i.e., given $x,y\in\{0,1\}^*$, we can decide in polynomial-time whether $x\prec y$ or $y\prec x$). *Then, for any set S such that $\mathtt{cnt}^{\prec}_S(x)\leq\exp(\mathrm{poly}(|x|))$, computing $\mathtt{cnt}^{\prec}_S$ is polynomial-time reducible to computing $\mathtt{enm}^{\prec}_S$.*

The condition $\mathtt{cnt}^{\prec}_S(x)\leq\exp(\mathrm{poly}(|x|))$ (equiv., $|\mathtt{cnt}^{\prec}_S(x)|\leq\mathrm{poly}(|x|)$) is obviously a necessary condition for efficiently computing $\mathtt{cnt}^{\prec}_S$ (regardless of the orcale we may use). We stress that this condition (which refers both to S and to $\prec$) is trivial in the case of the lexicographic order.

Proof: On input x, using oracle access to $\mathtt{enm}^{\prec}_S$ (and to $\prec$), we perform a binary search for the largest $i\in\mathbb{N}\cup\{0\}$ such that $\mathtt{enm}^{\prec}_S(i)\prec x$, while relying on $\mathtt{cnt}^{\prec}_S(x)\leq\exp(\mathrm{poly}(|x|))$. Once found, this i equals $\mathtt{cnt}^{\prec}_S(x)$, as desired. Details follow.

On input x, we maintain an interval of possible values for $\mathtt{cnt}^{\prec}_S(x)$, which is initiated to $[0,\exp(\mathrm{poly}(|x|))]$. We proceed in iterations such that in each iteration the size of the interval

shrinks by approximately a half. While the interval contains more than two integers, we pick an integer i in its (approximate) middle, check whether $\mathtt{enm}^{\prec}_S(i) \prec x$, and update the interval accordingly (i.e., if $\mathtt{enm}^{\prec}_S(i) \prec x$, then the lower bound of the interval is reset to i, and otherwise the upper bound is reset to $i-1$). Note that checking whether $\mathtt{enm}^{\prec}_S(i) \prec x$ is performed by using oracle access to $\mathtt{enm}^{\prec}_S$ and to $\prec$. Lastly, note that if $\mathtt{cnt}^{\prec}_S(x) \in \{i, i+1\}$, then $\mathtt{cnt}^{\prec}_S(x) = i+1$ if and only if $\mathtt{enm}^{\prec}_S(i+1) \prec x$. ■

Theorem 2.4 (typically, for sufficiently dense and efficiently recognizable S, computing $\mathtt{cnt}^{\prec}_S$ implies enumerating S): *Let $\prec$ be an order on strings such that the following* betweenness task[9] *can be solved in polynomial time: Given $x \prec y$, one should determine whether or not $y = \mathtt{succ}^{\prec}(x)$* (i.e., whether there exists z s.t. $x \prec z \prec y$)*, and in case $y \neq \mathtt{succ}^{\prec}(x)$ output a string z such that*

$$|\{w \in \{0,1\}^* : w \prec z\}| \;=\; \alpha \cdot |\{w \in \{0,1\}^* : w \prec x\}| \;+\; (1-\alpha) \cdot |\{w \in \{0,1\}^* : w \prec y\}|$$

for some $\alpha \in [0.1, 0.9]$.[10] *Then, for any polynomial-time recognizable set S such that $\mathtt{cnt}^{\prec}_S(x) \geq \exp(|x|^{\Omega(1)})$, computing $\mathtt{enm}^{\prec}_S$ is polynomial-time reducible to computing $\mathtt{cnt}^{\prec}_S$.*

The condition $\mathtt{cnt}^{\prec}_S(x) \geq \exp(|x|^{\Omega(1)})$ is shown to imply $|\mathtt{enm}^{\prec}_S(i)| \leq \mathrm{poly}(|i|)$, which is obviously a necessary condition. Note that, in case $\mathtt{succ}^{\prec}$ can be computed in polynomial time, deciding S is polynomial-time reducible to computing $\mathtt{cnt}^{\prec}_S$, because $x \in S$ if and only if $\mathtt{cnt}^{\prec}_S(x) < \mathtt{cnt}^{\prec}_S(\mathtt{succ}^{\prec}(x))$. Of course, in general, it is unlikely that computing $\mathtt{cnt}^{\prec}_S$ is polynomial-time reducible to deciding (see Remark 1.1).

Proof: We first show that the density condition implies that $|\mathtt{enm}^{\prec}_S(i)| \leq \ell(|i|)$, for some polynomial ℓ. Specifically, suppose that $\mathtt{cnt}^{\prec}_S(x) \geq \exp(|x|^{\epsilon})$, for some constant $\epsilon > 0$. Then, for any $i \in \mathbb{N}$, let $x \leftarrow \mathtt{enm}^{\prec}_S(i)$, and observe that $\mathtt{cnt}^{\prec}_S(x) = |\{z \in S : z \prec x\}| = i - 1 < i$. Hence, $\exp(|x|^{\epsilon}) \leq \mathtt{cnt}^{\prec}_S(x) < i$ implies $|x| < |i|^{1/\epsilon}$, and $|\mathtt{enm}^{\prec}_S(i)| < |i|^{1/\epsilon}$ follows. We now turn to the reduction itself.

On input $i \in \mathbb{N}$, using oracle access to S and to $\mathtt{cnt}^{\prec}_S$ (and an algorithm for the betweenness task), we perform a binary search for $x \in S$ such that $\mathtt{cnt}^{\prec}_S(x) = i - 1$, while observing that this x satisfies $|x| \leq \ell(|i|) \leq \mathrm{poly}(|i|)$. Once found, this x equals $\mathtt{enm}^{\prec}_S(i)$, as desired. Details follow.

On input i, we maintain a set of possible candidates for $\mathtt{enm}^{\prec}_S(i)$. This set is implicitly defined by two ("bounding") strings, denoted u and v, and equals $\{z : u \preceq z \preceq v\}$. Initially, we set u to be the first string according to $\prec$ (i.e., there exists no z s.t. $\mathtt{succ}^{\prec}(z) = u$), and $v = 0^{\ell(|i|)+1}$. We proceed in iterations such that in each iteration the number of candidates shrinks by a factor of at least 0.9. While $v \notin \{u, \mathtt{succ}^{\prec}(u)\}$, we use the "betweenness procedure" to pick string w in between u and v, obtain $\mathtt{cnt}^{\prec}_S(w)$, and update the foregoing bounding strings accordingly; that is, if $\mathtt{cnt}^{\prec}_S(w) \leq i - 1$, then we reset $u \leftarrow w$, and otherwise we reset to $v \leftarrow w$. If $u = v$, then $\mathtt{enm}^{\prec}_S(i) = u$ must hold, whereas if $v = \mathtt{succ}^{\prec}(u)$, then $\mathtt{enm}^{\prec}_S(i) \in \{u, v\}$ holds. In the latter case, we output u if $u \in S$, and output v otherwise. ■

Deriving Theorem 1.5 as a special case. As noted upfront, the conditions made in Theorems 2.3 and 2.4 regarding $\prec$ (i.e., being polynomial-time decidable, satisfying $\mathtt{cnt}^{\prec}_S(x) \leq \exp(\mathrm{poly}(|x|))$ for every S, and having an efficient betweenness procedure) hold for the lexicographic order. Hence, combining Theorems 2.3 and 2.4, we derive Theorem 1.5.

[9] Be warned that this is not the computational problem that is traditional called "betweenness" (cf. [3]).

[10] The slackness allowed here is rather arbitrary (as long as one avoids insisting on $\alpha = 1/2$). Note that, for every $i \leq j + 2$, it holds that $i < \alpha \cdot i + (1-\alpha) \cdot j < j$.

2.2 Additional enumeration problems

Continuing with the lexicographic order, we consider the lexicographic order on square Boolean matrices that is imposed by viewing the matrices as strings. Noting that the ordered set of square non-singular Boolean matrices, denoted S_{sns}, satisfies the conditions of Theorem 1.5, we infer that computing $\mathtt{enm}^{\mathtt{lex}}_{S_{\mathrm{sns}}}$ is polynomial-time equivalent to computing $\mathtt{cnt}^{\mathtt{lex}}_{S_{\mathrm{sns}}}$. We found it is somewhat easier to prove the following result in terms of $\mathtt{cnt}^{\mathtt{lex}}_{S_{\mathrm{sns}}}$.

Proposition 2.5 (efficiently enumerating square non-singular Boolean matrices): *The function $\mathtt{cnt}^{\mathtt{lex}}_{S_{\mathrm{sns}}}$ can be computed in polynomial time, where S_{sns} denote the set of square non-singular Boolean matrices.*

Proof: We mimic the standard procedure of selecting an n-by-n *non-singular* Boolean matrix uniformly at random. Recall that this process proceeds in iterations such that in the i^{th} iteration we select a row that is linearly independent of the previous $i-1$ rows. The number of such choices is $2^n - 2^{i-1}$, and the problem that we shall face soon is efficiently determining how many of these choices precede a given n-bit string. But before doing so, we note that the number of n-by-n non-singular matrices is

$$N_n \stackrel{\mathrm{def}}{=} \prod_{i\in[n]} \left(2^n - 2^{i-1}\right). \tag{2}$$

We now get to the aforementioned problem: For $k = i-1$, given a full rank k-by-n matrix G and a n-bit vector $x = x_1 \cdots x_n$, we ask how many of the vectors spanned by the rows of G precede x in lexicographic order. The key observation is that

$$|\{y\in\{0,1\}^k : yG \prec x\}| = \sum_{j\in[k]} |\{y\in\{0,1\}^k : (yG)_{[j-1]} = x_{[j-1]} \wedge (yG)_j < x_j\}|, \tag{3}$$

where $z_{[t]} = z_1 \cdots z_t$. The point is that j^{th} term in the r.h.s of Eq. (3) equals zero if $x_j = 0$ and otherwise it equals the number of solutions to a system of j linear equations (which are not necessarily independent or homogeneous).[11] Hence, we reduced our (intermediate) problem to determining the rank of matrices.

With the foregoing solution at hand, we determine the number of non-singular Boolean matrices that precede a given n-by-n matrix M as follows. For $i = 1, ..., n$, we let $G = G^{(i)}$ denote the matrix that represents the first $i-1$ rows of M, and let $x = x^{(i)}$ denote the i^{th} row of M. Now, we compute the number of the vectors spanned by the rows of G that precede x in lexicographic order, denoting this number by $\mathtt{N}^{(i)}(M)$. We claim that *if M is non-singular, then the number of n-by-n non-singular matrices that precede it equals*

$$\mathtt{N}(M) \stackrel{\mathrm{def}}{=} \sum_{i\in[n]} \mathtt{N}^{(i)}(M) \cdot \prod_{j\in[i+1,n]} (2^n - 2^{j-1}), \tag{4}$$

The claim follows by observing that an n-by-n non-singular matrix M' precedes M if for some $i \in [n]$ the first $i-1$ rows of these two matrices agree while the i^{th} row of M' precedes the i^{th} row of M. Note that the remaining $n-i$ rows of M' can be any of the allowed $\prod_{j\in[i+1,n]}(2^n - 2^{j-1})$ possibilities. Hence, for an n-by-n non-singular matrix M, it holds that $\mathtt{cnt}^{\mathtt{lex}}_{S_{\mathrm{sns}}}(M) = \sum_{k\in[n-1]} N_k + \mathtt{N}(M)$.

[11] Recalling that G and x are fixed, the actual variables are the bits of y. For $\ell \in [j-1]$, the ℓ^{th} equation is $(yG)_\ell = x_\ell$, and the j^{th} equation is $(yG)_j = 0$.

In the general case (i.e., when M may be singular), we let $r(M) \in \{0,1,2...,n\}$ denote the maximal number r such that the first r rows of M are independent, and return 0 if $r(M) = 0$. Otherwise (i.e., $r(M) \in [n]$), we stop the foregoing process after performing iteration $i = r(M)$; that is, we output $\sum_{k\in[n-1]} N_k + \mathtt{N}(M)$ such that

$$\mathtt{N}(M) \stackrel{\text{def}}{=} \sum_{i\in[r(M)]} \mathtt{N}^{(i)}(M) \cdot \prod_{j\in[i+1,n]} (2^n - 2^{j-1}). \tag{5}$$

Indeed, Eq. (5) differens from Eq. (4) only in the scope of the summation; that is, Eq. (4) is the special case of Eq. (5) that corresponds to $r(M) = n$.

A different way of presenting the general case amounts to saying that we replace M by the first (in lexicographic order) non-singular n-by-n matrix that agrees with M on the first $r(M)$ rows, and invoke the special case on the resulting matrix (i.e. denoting the resulting matrix by R, it holds that $\mathtt{cnt}^{\mathtt{lex}}_{S_{\mathrm{sns}}}(M) = \mathtt{cnt}^{\mathtt{lex}}_{S_{\mathrm{sns}}}(R)$). ■

Generalization to other finite fields. The proof of Proposition 2.5 extends to any finite field, except that handling the j^{th} term in the r.h.s of Eq. (3) requires considering all values smaller than x_j. Hence, the complexity of enumeration grows linearly with the size of the finite field. Actually, we can get complexity that is linear in the characteristic of the finite field, but it is unclear whether one can do better than that.

On enumerating finite sequences of natural numbers. The set of t-tuples of natural numbers (considered in Observation 1.3) is quite a popular countable set. Another popular countable set is the set of all finite sequences of natural numbers. Recall that we defined the standard order on t-tuples of natural numbers such that the dominant term is the sum of the numbers in the tuple; that is, $(a_1,...,a_t) \in \mathbb{N}^t$ precedes $(b_1,...,b_t)$ whenever $\sum_{i\in[t]} a_i < \sum_{i\in[t]} b_i$. Applying the same principle of sequences of varying number of elements does not seem natural, because it would mean that the n-long sequence of 1's (resp., any $n/2$-long sequence over $\{1,2\}$) precedes the sequence $(1,n)$. Indeed, the foregoing suggestion fails to account for the length of the sequences. In contrast, we argue that the order presented in the next result is the natural one in the current context.

Proposition 2.6 (efficiently enumerating finite sequences of natural numbers according to the length of the sequences): *Consider the set, denoted $\mathbb{N}^+ = \bigcup_{t\in\mathbb{N}} \mathbb{N}^t$, of finite sequences of natural numbers with respect to the following order $\prec$ that places $(m_1,...,m_{t'})$ before $(n_1,...,n_{t''})$ if one of the following conditions hold.*

1. *$\sum_{i\in[t']} |m_i| < \sum_{i\in[t'']} |n_i|$, where as usual $|n| = \lceil \log_2(n+1) \rceil$.*
2. *$\sum_{i\in[t']} |m_i| = \sum_{i\in[t'']} |n_i|$ and $t' < t''$.*
3. *$t \stackrel{\text{def}}{=} t' = t''$ and $\sum_{i\in[t]} |m_i| = \sum_{i\in[t]} |n_i|$ and $(|m_1|,...,|m_t|)$ precedes $(|n_1|,...,|n_t|)$ according to the standard order of t-tuples over $\mathbb{N}$.*
4. *$(|m_1|,...,|m_{t'}|) = (|n_1|,...,|n_{t''}|)$ and for some $k \in [t'-1]$ it holds that $m_k < n_k$ and $m_i = n_i$ for all $i \in [k-1]$.*

Then, the foregoing ordered set can be enumerated in polynomial time.

The foregoing order on sequences of varying length induces an alternative order on $\mathbb{N}^t$. As we shall see in the following proof, it is easier to enumerate $\mathbb{N}^t$ according to this alternative order than to do so according to the standard order (which is considered in Observation 1.3).

Proof: Again, using Theorem 2.4, we focus on computing the corresponding counting function $\mathtt{cnt}^{\prec}_{\mathbb{N}^+}$. Given a sequence $\overline{a} = (a_1, ..., a_t) \in \mathbb{N}^+$, we proceed as follows.

- Letting $\ell = \sum_{i\in[t]} |a_i|$, for each $\ell' \in [\ell - 1]$ and $t' \in [\ell']$, we compute the number of sequences $(\ell_1, ..., \ell_{t'}) \in \mathbb{N}^{t'}$ such that $\sum_{i\in[t']} \ell_i = \ell'$. Recalling that this number equals $N_{\ell',t'} \stackrel{\text{def}}{=} \binom{\ell'-1}{t'-1}$, we can compute all these numbers (i.e., all $N_{\ell',t'}$'s for $\ell' \in [\ell-1]$ and $t' \in [\ell']$) in poly(ℓ)-time.

  Furthermore, For each $\ell' \in [\ell - 1]$ and $t' \in [\ell']$, the number of sequences $(b_1, ..., b_{t'}) \in \mathbb{N}^+$ such that $\sum_{i\in[t']} |b_i| = \ell'$ equals $N_{\ell',t'} \cdot 2^{\ell'-t'}$, because the number of sequences $(b_1, ..., b_{t'})$ that satisfy $|b_i| = \ell_i$ (equiv., $b_i \in [2^{\ell_i-1}, 2^{\ell_i} - 1]$) for each $i \in [t']$ equals $\prod_{i\in[t']} 2^{\ell_i-1} = 2^{\ell'-t'}$.

  Hence, we can efficiently compute the number of sequences that precede $\overline{a}$ per Condition 1, which equals
  $$\sum_{\ell'\in[\ell-1]} \sum_{t'\in[\ell']} N_{\ell',t'} \cdot 2^{\ell'-t'}.$$

- Similarly, we can efficiently compute the number of sequences that precede $\overline{a}$ per Condition 2. Specifically, the number of sequences that precede $\overline{a}$ per Condition 2 is $\sum_{t'\in[t-1]} N_{\ell,t'} \cdot 2^{\ell-t'}$, where $\ell = \sum_{i\in[t]} |a_i|$.

- Turning to Condition 3, recall that an efficient algorithm for computing the number of sequences $(\ell_1, ..., \ell_t)$ that precede $(|a_1|, ..., |a_t|)$ was outlined in Observation 1.3 (see Footnote 7). The number of sequences that precede $\overline{a}$ per Condition 3 is $2^{\ell-t}$ times larger.

- Lastly, we compute the number of sequences that precede $\overline{a}$ per Condition 4; that is, we compute the number of $(b_1, ..., b_{t'}) \in \mathbb{N}^t$ such that $(|b_1|, ..., |b_t|) = (|a_1|, ..., |a_t|)$ and for some $k \in [t - 1]$ it holds that $b_k < a_k$ and $b_i = a_i$ for all $i \in [k - 1]$. For each $k \in [t - 1]$, the corresponding number is $(a_k - 1) \cdot 2^{\ell-\sum_{i\in[k]} |a_i|}$.

Having established the fact that $\mathtt{cnt}^{\prec}_{\mathbb{N}^+}$ can be computed in polynomial-time and wishing to invoke Theorem 2.4, we need to verify that the betweenness task can be performed in polynomial-time. Unlike in prior cases, this is non-trivial, due to the complexity of the order involved. Given $(m_1, ..., m_{t'}) \prec (n_1, ..., n_{t''})$, we solve the "between task" by considering the following two cases.

***The case of equal length-sequences*:** $(|m_1|, ..., |m_{t'}|) = (|n_1|, ..., |n_{t''}|)$.

Letting $t \stackrel{\text{def}}{=} t' = t''$ and $\ell_i \stackrel{\text{def}}{=} |m_i| = |n_i|$ for every $i \in [t]$, we identify $k \in [t - 1]$ such that $m_k < n_k$ and $m_j = n_j$ for every $j \in [k-1]$. If $m_k \leq n_k - 2$, then $(m_1, ..., m_{k-1}, i_k, i_{k+1}, ..., i_t)$ such that $i_k = \lfloor (m_k + n_k)/2 \rfloor$ and $i_j = \lceil 0.75 \cdot (2^{\ell_j} - 1) \rceil$ for every $j \in [k + 1, t]$ is an adequate answer.[12] Otherwise (i.e., $m_k = n_k - 1$), a more careful examination of the sequences is required.

[12] The key observation is that, for every v_k, the number of $(t - k)$-tuples $(v_{k+1}, ..., v_t) \in \mathbb{N}^{t-k}$ such that $|v_j| = \ell_j$ (equiv., $v_j \in [2^{\ell_j-1}, 2^{\ell_j} - 1]$) equals $\prod_{j=k+1}^{t} 2^{\ell_j-1} = 2^{\ell'-(t-k)}$, where $\ell' = \sum j = k + 1^t \ell_j$. The argument then focuses on the worst case in which $m_j = 2^{\ell_j-1}$ and $n_j = 2^{\ell_j} - 1$ for every $j \in [k + 1, t]$.

***The case of different length-sequences*:** $(|m_1|,...,|m_{t'}|) \neq (|n_1|,...,|n_{t''}|)$.

In this case, we observe that

$$(m_1,...,m_{t'}) \preceq (2^{|m_1|}-1,...,2^{|m_{t'}|}-1) \prec (2^{|n_1|-1},...,2^{|n_{t''}|-1}) \preceq (n_1,...,n_{t''}).$$

Computing the value of $\mathtt{cnt}^{\prec}_{\mathbb{N}^+}$ on these four sequences, we determine whether one of the two intermediate sequences can be used as a valid answer. Otherwise, a valid answer should appear inbetween one of the three adjacent pairs. The first and third subcases are handled by the previous case, and so we are left with solving the betweenness problem for the second subcase (i.e., $(2^{|m_1|}-1,...,2^{|m_{t'}|}-1) \prec (2^{|n_1|-1},...,2^{|n_{t''}|-1})$).

Thus, given $(\ell'_1,...,\ell'_r)$ and $(\ell''_1,...,\ell''_s)$ such that $(2^{\ell'_1}-1,...,2^{\ell'_r}-1) \prec (2^{\ell''_1-1},...,2^{\ell''_s-1})$, we have to find $(w_1,...,w_t)$ that satisfies the betweenness condition. Letting $\ell' \stackrel{\text{def}}{=} \sum_{i\in[r]} \ell'_i$ and $\ell'' \stackrel{\text{def}}{=} \sum_{i\in[s]} \ell''_i$, this is done by considering all possible $\ell \in [\ell',\ell'']$ and all possible $t \in [\ell]$ and $(p,q) \in [t] \times [\ell]$, which correspond to Conditions 1–3, respectively. The key observation is that the number of possibilities is polynomial in ℓ'', which is the length of our original input (i.e., the sequences $(m_1,...,m_{t'})$ and $(n_1,...,n_{t''})$), and for each possibility the relevant computations are easy.

Indeed, the tedious details were omitted here. ■

3 The Amortized Notion: More Details

In this section, we formulate more rigorously the notions that are mentioned in Section 1.2

Whereas Definition 2.1 refers to the complexity of finding the i^{th} element in a set S, here we consider the complexity of producing the list of the first i elements. This notion yield a notion of amortized complexity, but one in which information generated for producing the first $i-1$ elements can be used in producing the i^{th} element. A more stringent notion may refer to the average time it takes to evaluate $\mathtt{enm}^{\prec}_S$ on the first i numbers (see Secton 4).

Definition 3.1 (the amortized complexity of enumeration): *For an infinite countable set $S \subseteq \{0,1\}^*$ and an order on strings, denoted $\prec$, let $\mathtt{enm}^{\prec}_S$ be as in Definition 2.1. Then, the* amortized complexity of enumerating *S is defines as $T_S(i)/i$, where $T_S(i)$ is the total time it takes to produce the first i elements in S; that is, we consider the time complexity of algorithms that, given $i \in \mathbb{N}$, output the sequence $(\mathtt{enm}^{\prec}_S(1),...,\mathtt{enm}^{\prec}_S(i))$.*

Note that the amortized complexity of enumerating S is polynomial if and only if $T_S(i) = \widetilde{O}(i) = i \cdot \text{poly}(|i|)$. We also observe that if the "successive function of S w.r.t $\prec$" (i.e., the mapping $\mathtt{enm}^{\prec}_S(i) \mapsto \mathtt{enm}^{\prec}_S(i+1)$) can be computed in polynomial-time and $|\mathtt{enm}^{\prec}_S(i)| \leq \text{poly}(|i|)$, then the amortized complexity of enumerating S is polynomial. The corresponding algorithm works by iterative invocations of the successive function (of S w.r.t $\prec$), and makes a modest requirement regarding the density of S (i.e., $|\mathtt{enm}^{\prec}_S(i)| \leq \text{poly}(|i|)$ implies that for some constant $\epsilon > 0$ it holds that $|S \cap \bigcup_{i\in[n]}\{0,1\}^i| \geq 2^{n^\epsilon}$). A different algorithm is used for the following result.

Theorem 3.2 (the amortized complexity of enumerating dense and efficiently recognizable sets): *For an infinite countable set $S \subseteq \{0,1\}^*$ and the lexicographic order on strings, suppose that S is polynomial-time recognizable and that $|S \cap \{0,1\}^n| \geq 2^n/\text{poly}(n)$ (for every $n \in \mathbb{N}$). Then, the amortized complexity of enumerating S is polynomial.*

For simplicity, we have restricted ourselves to the lexicographic order on strings, but any order $\prec$ that is monotonically non-decreasing with the length of strings (i.e., $x \prec y$ implies $|x| \leq |y|$) will do. A more general statement appears in Theorem 3.4.

Proof: We consider an algorithm that, on input $i \in \mathbb{N}$, generates the first $\text{poly}(|i|) \cdot 2^{|i|}$ strings (in lexicographic order), determines which of these strings is in S, and outputs the i first strings that are in S. Correctness follows by the density guarantee, which implies that the initial list contains more than i elements of S (i.e., for $n = |i| + O(\log |i|)$, we have $|S \cap \{0,1\}^n| \geq 2^n/\text{poly}(n)$, which is greater than $2^{|i|}$). The total time spend by this algorithm is $\text{poly}(|i|) \cdot 2^{|i|} \cdot \text{poly}(|i|) = \widetilde{O}(i)$, which implies that the amortized complexity is $\text{poly}(|i|)$. ■

Corollary 3.3 (the amortized complexity of enumerating the primes and the rationals): *The amortized complexity of enumerating the following two sets is polynomial.*

1. *The set of all primes* (ordered according to their size).
2. *The set of all positive rational numbers* (represented by ordered pairs of natural numbers that are ordered as in Observation 1.3).

Proof: Item 1 follows by invoking Theorem 3.2, while recalling that the number of n-bit long primes is $\Omega(2^n/n)$ and using the primality tester of [1]. To establish Item 2, we need to show that the number of rationals that are represented by pairs (a,b) such that $|(a,b)| = n$ (and $\gcd(a,b) = 1$) is at least $2^n/\text{poly}(n)$. This follows even by considering only pairs of the form $(p,1)$ such that p is a prime. ■

Theorem 3.4 (a generalization of Theorem 3.2): *Suppose that $\prec$ is an order on strings such that the amortized complexity of enumerating $U \stackrel{\text{def}}{=} \{0,1\}^*$ according to order $\prec$ is polynomial. For an infinite countable set $S \subseteq \{0,1\}^*$, suppose that S is polynomial-time recognizable and that $|S \cap \{\mathtt{enm}_U^{\prec}(i) : i \in [2^n]\}| \geq 2^n/\text{poly}(n)$ (for every $n \in \mathbb{N}$). Then, the amortized complexity of enumerating S according to $\prec$ is polynomial.*

The restriction to orders that allow for an efficient amortized enumeration of all strings is essential. Violations of this restriction may be due to the order having huge gaps; specifically, it may be that for a fast growing function $f : \mathbb{N} \to \mathbb{N}$ there are infinitely many i's such that $|\mathtt{enm}_U^{\prec}(i)| \geq f(i)$.[13]

Proof: We follow the strategy of the proof of Theorem 3.2, except that here we generate the first $\text{poly}(|i|) \cdot 2^{|i|}$ strings according to the given order. Correctness follows by relying on the modified density guarantee, which refers to the given order. ■

Transitivity. By a straightforward generalization of the proof of Theorem 3.4, we obtain the following.

Theorem 3.5 (a generalization of Theorem 3.4): *For infinite countable sets S, T and an order $\prec$ on T, suppose that the amortized complexity of enumerating T according to $\prec$ is polynomial. If S is polynomial-time recognizable and $|S \cap \{\mathtt{enm}_T^{\prec}(i) : i \in [2^n]\}| \geq 2^n/\text{poly}(n)$ (for every $n \in \mathbb{N}$), then the amortized complexity of enumerating S according to $\prec$ is polynomial.*

[13] Consider, for example, for any monotonically increasing function $g : \mathbb{N} \to \mathbb{N}$, an order derived from the standard lexicographic order by moving $0^{g(n)+1}$ from its standard location (right after $1^{g(n)}$) to the location right after 0^{n+1}).

Note that T is *not* necessarily dense with respect to U (i.e., $|T \cap \{\mathtt{enm}_U^{\prec}(i) : i \in [2^n]\}| \geq 2^n/\mathrm{poly}(n)$ does not necessarily hold). Consider, for example, the lexicographic order and the set $T = \{x0^{|x|} : x \in T'\}$ such that the amortized complexity of enumerating T' is polynomial (e.g., T' is the set of primes).

Some "degrees of freedom". We observe that in the current context, when given $i \in \mathbb{N}$, the ability to generate a list that contains the i first elements of the ordered set suffices.

Observation 3.6 (implicit in [8]): *Let $\prec$ be a polynomial-time decidable order on strings* (i.e., given $x, y \in \{0,1\}^*$, we can decide in polynomial-time whether $x \prec y$ or $y \prec x$). *Suppose that the set S is polynomial-time recognizable and there exists an algorithm that, given $i \in \mathbb{N}$, runs for $t = \widetilde{O}(i)$ steps and outputs a list that contains the i first elements of S. Then, the amortized complexity of enumerating S according to $\prec$ is polynomial. The key observation is that, using the hypothesis, we can recognize the elements of S in the list, and sort the list according to $\prec$, all within $\widetilde{O}(t)$-time.*

4 An Average Case Notion

In continuation to brief comments made in the introduction and in Section 3, we present a definition of average case complexity of enumeration problems. Following Levin [9] (see [5, Apdx]), we avoid the naive formulation, and present instead the following one.

Definition 4.1 (average case complexity of enumeration): *Let $c_A : \mathbb{N}^* \to \mathbb{N}^*$ be an arbitrary notion of complexity of algorithm A such that $c_A(i)$ is a function of the number of steps that A takes on input i* (e.g., $c_A(i)$ may be a square root of the number of steps).

- *For $(S, \prec)$ and $\mathtt{enm}_S^{\prec}$ as in Definition 3.1, we say that A* enumerates *S* within average complexity *C if $\sum_{i \in [n]} c_A(i)/n \leq C(|n|)$.*
- *Letting $t_A(i)$ denote the number of steps taken by A on input i, we say that A* enumerates *S* in average polynomial time *if for some polynomial p it holds that $t_A(i) \leq p(c_A(i))$ and $\sum_{i \in [n]} c_A(i)/n \leq O(|n|)$.*

The naive formulation being avoided uses $c_A = t_A$ and $C = p$; that is, it says that a function is polynomial on the average if its average is polynomial. Instead, we say that a function is polynomial on the average if it is polynomial in a function that is linear on the average. The point is that the alternative formulation is closed under polynomial composition, whereas the naive one is not. Actually, we believe that the notion of *typically polynomial-time* is more appealing than the notion of *average polynomial-time* (see [6, Sec. 10.2.1.1]).

Definition 4.2 (enumeration in typical polynomial-time): *For $(S, \prec)$ and t_A as in Definition 4.1, we say that A* enumerates *S* in typical polynomial time *if for some polynomial p and some negligible*[14] *function μ, for all n's it holds that $|\{i \in [n] : t_A(i) > p(|i|)\}| \leq \mu(|n|) \cdot n$.*

[14] A function $\mu : \mathbb{N} \to [0,1]$ is called negligible if for every positive polynomial q and all sufficiently large ℓ it holds that $\mu(\ell) < 1/q(\ell)$.

Obviously the strict notion of polynomial-time implies the average (resp., typical) notion, which in turn implies the amortized notion. An evidence for the separation between the average (resp., typical) he amortized notions is provided by the problems of enumerating primes and rational numbers. Recall that these two ordered sets are enumerated in amortized polynomial-time (Corollary 3.3), whereas the difficulties in enumerating them in the strict sense (cf. Observation 1.2 and 1.4) extend also to the average (resp., typical) sense. Specifically, the issue is not finding an object of an approximate desired size, which is typically easy, but rather finding an object of a specific location in the order. For example, given N, it is "typically" easy to find a prime in the interval $[N, N + \mathrm{poly}(\log N)]$, but it is typically hard to determine the number of primes that are smaller than N (equiv., find the i^{th} prime, when given i).[15]

As for a separation between the strict notion of polynomial-time and the average (resp., typical) notion, we provide evidence to it by using a construction akin to Remark 1.1

Observation 4.3 (strict vs average (resp., typical) polynomial-time enumeration): *Assuming that $\#\mathcal{P}$-complete problems cannot be solved in polynomial-time, there exists a set $S \subset \{0,1\}^*$ that cannot be enumerated in strict polynomial-time, but can be enumerate in typical and average polynomial-time, where in all cases the enumeration is according to the lexicographic order.*

Proof: Let $R \subseteq \{0,1\}^* \to \{0,1\}^*$ be a $\#\mathcal{P}$-complete search problem and let $R(x) = \{y : (x,y) \in R\}$. Note that we may assume, without loss of generality, that $y \in R(x)$ implies that $|y| = \ell(|x|)$ for some polynomial $\ell : \mathbb{N} \to \mathbb{N}$. Let $\chi_{x,y} = 1$ if $(x,y) \in R$ and $\chi_{x,y} = 0$ otherwise. Consider the set $S = S' \cup S''$ such that

$$\begin{aligned}
S' &\stackrel{\text{def}}{=} \left\{x0^{\ell(|x|)}1y : y \in R(x)\right\} \cup \left\{x0^{\ell(|x|)}0y : y \in \left(\{0,1\}^{\ell(|x|)} \setminus R(x)\right)\right\} \\
&= \left\{x0^{\ell(|x|)}\chi_{x,y}y : y \in \{0,1\}^{\ell(|x|)}\right\} \\
S'' &\stackrel{\text{def}}{=} \left\{xz'z'' \in \{0,1\}^* : z' \in \left(\{0,1\}^{\ell(|x|)} \setminus \{0^{\ell(|x|)}\}\right) \wedge z'' \in \{0,1\}^{\ell(|x|)+1}\right\}
\end{aligned}$$

Note that $w \in S$ implies that $|w| = n + 2\ell(n) + 1$ for some $n \in \mathbb{N}$. Furthermore, for $w = xz'z'' \in \{0,1\}^{n+2\ell(n)+1}$ such that $|x| = n$ and $|z'| = \ell(n)$, either $z' \neq 0^{\ell(n)}$ and $z'' \in \{0,1\}^{\ell(n)+1}$ or $z' = 0^{\ell(n)}$ and $z'' = \sigma y$ such that $y \in R(x)$ if and only if $\sigma = 1$.

Note that computing the number of R-solutions for x (i.e., $|R(x)|$) reduces to computing $\mathtt{cnt}_S^{\prec}$. Specifically,

$$|R(x)| \;=\; \mathtt{cnt}_S^{\prec}(x0^{\ell(|x|)-1}10^{\ell(|x|)+1}) - \mathtt{cnt}_S^{\prec}(x0^{\ell(|x|)}10^{\ell(|x|)})$$

because

$$\begin{aligned}
&\mathtt{cnt}_S^{\prec}(x0^{\ell(|x|)-1}10^{\ell(|x|)+1}) - \mathtt{cnt}_S^{\prec}(x0^{\ell(|x|)}10^{\ell(|x|)}) \\
&= \left|\left\{z \in \{0,1\}^{2\ell(|x|)+1} : z \prec \mathtt{succ}^{\prec}(0^{\ell(|x|)}11^{\ell(|x|)})\right\}\right| - \left|\left\{z \in \{0,1\}^{2\ell(|x|)+1} : z \prec 0^{\ell(|x|)}10^{\ell(|x|)})\right\}\right| \\
&= \left|\left\{0^{\ell(|x|)}1y : y \in R(x)\right\}\right|
\end{aligned}$$

where $\mathtt{succ}^{\prec}(w)$ denotes the string succeeding w according to $\prec$.

[15] The first assertion (i.e., ease of finding primes) is a bit inaccurate. The prime number theorem only implies that, for any constant $c > 0$, the fraction of n-bit numbers N such that $[N, N + n^{c+1}]$ contains no prime is at most n^{-c}. Recall, however, that Cramer's conjecture is that $[N, N + n^2]$ always contains a prime.

On the other hand, the density of S' within all strings in $S \cap \{0,1\}^{n+2\ell(n)+1}$ is exactly $2^{-(\ell(n)+1)}$, whereas counting the number of strings that precede a given string in S'' is easy. This is the case because the number of strings in $S' \cap \{0,1\}^{n+2\ell(n)+1}$ is exactly $2^{\ell(n)}$. Hence, given $i \in \mathbb{N}$ such that $\mathtt{enm}^{\prec}_S(i) \in S''$, we can compute $\mathtt{enm}^{\prec}_S(i)$ in polynomial time. Lastly, note that $(n+2\ell(n)+1)$-bit long strings of S' can be enumerated in time $2^{n+\ell(n)} \cdot \mathrm{poly}(n)$. Thus, S can be enumerated in (typical and) average polynomial-time. ■

5 Open Problems (a compilation)

In this section, we compile a list of the open problem that are explicit or implicit in the prior sections. We focus on problems related to the strict notion of enumeration.

Recall that, by Theorem 1.5, enumerating prime numbers is computationally equivalent to counting the number of primes that are smaller than a given natural number. The latter problem was studied in the number theoretic community, and the best known algorithm for the latter problem runs in exponential-time: on input an n-bit number, the algorithm runs in $\widetilde{O}(2^{n/2})$-time (see [7] and references therein). Putting aside the obvious question of whether this can be improved, we wonder if some additional evidence can be given for the perceived difficulty of the problem.

Open Problem 5.1 (the complexity of strictly enumerating the prime numbers): *Can one provide complexity theoretic justification for the perceived difficulty of strictly enumerating the prime numbers?*

Turning to the other classical problem (i.e., enumeration of rational numbers), we ask the obvious question (see next title) as well as questions regarding the relative complexity of sub-tasks.

Open Problem 5.2 (the complexity of strictly enumerating the rational numbers): *For* $\mathbb{Q} = \{(a,b) \in \mathbb{N}^2 : \gcd(a,b) = 1\}$ *and* $\mathtt{enm}^{\mathtt{lex}}_{\mathbb{Q}}$*, when defined according to the lexicographic order on these pairs* (i.e., (a,b) appears before (a',b') if either $a+b < a'+b'$ or $a+b = a'+b'$ and $a < a'$)*, what is the complexity of computing* $\mathtt{enm}^{\mathtt{lex}}_{\mathbb{Q}}$*? Recall that, by Observation 1.4, computing* $\mathtt{enm}^{\mathtt{lex}}_{\mathbb{Q}}$ *is at least as hard as computing* $s \mapsto \sum_{i \in [s]} \varphi(i)$*, which is at least as hard as factoring integers. Are any of these three problem computationally equivalent?*

A positive answer regarding the first two problems would follow if, given $s \in \mathbb{N}$ and $a \in [s-1]$ *along with the factorization of* s, one can efficiently determine the size of the set $|\{a' \in [a-1] : \gcd(a',s) = 1\}|$. This is the case because of the following facts:

1. The number of pairs in $\mathbb{Q}$ that precede (a,b) equals
$$\left(\sum_{i \in [a+b-1]} \varphi(i) \right) + \left| \{a' \in [a-1] : \gcd(a', a+b) = 1\} \right| . \tag{6}$$
2. Computing $s \mapsto \sum_{i \in [s]} \varphi(i)$ allows to compute the main sum in Eq. (6) as well as to compute the function φ.
3. Computing φ allows to factor integers [11, Sec. 10.4].

Hence, we pose the following problem.

Open Problem 5.3 (the complexity of determining the size of a prefix of the multiplicative group modulo a factored composite): *Given* $((p_1, e_1), ..., (p_t, e_t))$ *and* $r \in [s-1]$, *where* $s = \prod_{i \in [t]} p_i^{e_i}$, *can we efficiently determine the size of the set* $|\{m \in [r] : (\forall i \in [t])\ \gcd(m, p_i) = 1\}|$*?*

We note that the special case in which the e_i's are all 1 is computationally equivalent to the general case.

We now turn to another natural enumeration problem (i.e., enumerating non-singular square matrices). Recall that the proof of Proposition 2.5 extends to any finite field, except that the complexity may be linear in the characteristic of the finite field. Recall that difficulty is in evaluating the quantity in Eq. (3). This leads to the following problem.

Open Problem 5.4 (the complexity of strictly enumerating square non-singular matrices over arbitrary finite fields) *Can the complexity of enumerating square non-singular matrices over a finite field be polylogarithmic in the size of the field? In particular, can the following problem be solved in* $\mathrm{poly}(n \log p)$*-time? Given a full rank* k*-by-*n *matrix* G *over* $\mathrm{GF}(p)$*, where* $k < n$ *and* p *is a prime number, and a vector* $x \in \mathrm{GF}(p)^n$*, determine the number of the vectors spanned by the rows of* G *that precede* x *in lexicographic order.*

Computationally tractable orders on strings. The general results stated in Section 2 impose three minimal restrictions on the order $\prec$ (hinted at by the word "typically" in the titles of Theorems 2.2–2.4). These restrictions (or conditions), which are briefly discussed next, assert the tractability of natural computational problems regarding orders on strings.

***Efficient decidability*:** There exists a polynomial-time algorithm that, on input x and y, determines whether or not $x \prec y$.

***Weak conformity with length* :** If $x \prec y$, then $|x| \leq \mathrm{poly}(|y|)$.

This may be viewed as asserting that, on input x, one can efficiently find a string y such that $x \prec y$.

(A much more stringent requirement may postulate that $x \prec y$ implies $|x| \leq |y|$.)

***An efficient betweenness procedure*:** For starters, we require an efficient decision procedure that, on input x and y, determines whether or not $y = \mathtt{succ}^{\prec}(x)$, where $\mathtt{succ}^{\prec}(x)$ is the successor of x with respect to $\prec$.

Furthermore, letting $\mathtt{idx}^{\prec}(v) \stackrel{\mathrm{def}}{=} |\{w \in \{0,1\}^* : w \prec v\}| + 1$, we require an efficient algorithm that, given x and y such that $\mathtt{succ}^{\prec}(x) \prec y$, outputs a string z such that for some $\beta \in [0.1, 0.9]$ it holds that $\mathtt{idx}^{\prec}(z) = \mathtt{idx}^{\prec}(x) + \beta \cdot (\mathtt{idx}^{\prec}(y) - \mathtt{idx}^{\prec}(x))$.

Note that the last two conditions imply that $\mathtt{succ}^{\prec}$ can be computed in polynomial-time.[16] In contrast, even assuming that $x \prec y$ implies $|x| \leq |y|$ (and that $\prec$ can be efficiently decided), does not allow to reduce the betweenness task to $\mathtt{succ}^{\prec}$.

We wounder whether these three conditions capture what one may consider as the class of "computationally tractable orders on the set of strings" and what one can say about this class.

[16] The claim follow by observing that, given x and y such that $x \prec y$, we can find $\mathtt{succ}^{\prec}(x)$ by iteratively replacing y with a string y' that is between x and y. Letting $\ell(|x|) \geq |x|$ be an upper-bound on the length of strings that precede x according to $\prec$, we observe that $x \prec 1^{\ell(|x|)+1}$ must hold, and so we can start the foregoing process with $y \leftarrow 1^{\ell(|x|)+1}$.

Acknowledgments

I am grateful to Madhu Sudan for crucial help with the proof of Proposition 2.5.

References

[1] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics*, Vol. 160 (2), pages 781–793, 2004.

[2] N. Calkin and H.S. Wilf. Recounting the Rationals. *American Math. Monthly*, Vol. 107 (4), pages 360–363, 2000.

[3] B. Chor and M. Sudan. A geometric approach to betweenness. *SIAM J. on Disc. Math.*, Vol. 11 (4), pages 511–523, 1998.

[4] N. Creignou, M. Kroll, R. Pichler, S. Skritek, and H. Vollmer. A Complexity Theory for Hard Enumeration Problems. *Discrete Applied Mathematics*, Vol. 268, pages 191–209, 2019.

[5] O. Goldreich. Notes on Levin's Theory of Average-Case Complexity. *ECCC*, TR97-058, Dec. 1997.

[6] O. Goldreich. *Computational Complexity: A Conceptual Perspective.* Cambridge University Press, 2008.

[7] D. Hirsch, I. Kessler, abd U. Mendlovic. Computing $\pi(N)$: An elementary approach in $\widetilde{O}(N^{1/2})$ time. `arXiv:2212.09857` [math.NT], 2022.

[8] D.S. Johnson, C.H. Papadimitriou, and M. Yannakakis. On generating all maximal independent sets. *Inf. Process. Lett.*, Vol. 27 (3), pages 119–123, 1988.

[9] L.A. Levin. Average Case Complete Problems. *SIAM J. on Comput.*, Vol. 15, pages 285–286, 1986.

[10] D.E. Knuth. *The Art of Computer Programming, Volume 4A.* Addison-Wesley Professional, 2011.

[11] V. Shoup. *A Computational Introduction to Number Theory and Algebra.* Cambridge University Press, 2008.