The Algorithmics Column

by

Thomas Erlebach

Department of Informatics
University of Leicester
University Road, Leicester, LE1 7RH.
t.erlebach@leicester.ac.uk

Enumeration Complexity

Yann Strozecki
Université de Versailles
yann.strozecki@uvsq.fr

Abstract

An enumeration problem is the task of listing a set of elements without redundancies, usually the solutions of some combinatorial problem. The enumeration of cycles in a graph appeared already 50 years ago [96], while fundamental complexity notions for enumeration have been proposed 30 years ago by Johnson, Yannakakis and Papadimitriou [65]. Nowadays several research communities are working on enumeration from different point of views: graph algorithms, parametrized complexity, exact exponential algorithms, logic, database, enumerative combinatorics, applied algorithms to bioinformatics, cheminformatics, networks ...

In the last ten years, the topic has attracted more attention and these different communities began to share their ideas and problems, as exemplified by two recent Dagstuhl workshops "Algorithmic Enumeration: Output-sensitive, Input-Sensitive, Parameterized, Approximative" and "Enumeration in Data Management" or the creation of Wikipedia pages for enumeration complexity and algorithms.

In this column, we focus on the structural complexity of enumeration, trying to capture different notions of tractability. The beautiful algorithmic methods used to solve enumeration problems are only briefly mentioned when relevant and would require another column. Much of what is presented here is inspired by several PhD theses and articles [94, 17, 78, 77], in particular a good part of this text is borrowed from [21, 20].

1 Introduction

Modern enumeration algorithms date back to the 70's with graphs algorithms but older problems can be reinterpreted as enumeration: the baguenaudier game [76] from the 19th century can be seen as the problem of enumerating integers in Gray code order. There are even thousand years old examples of methods to list simple combinatorial structures, the subsets or the partitions of a finite set, as reported by Ruskey [92] in his book on combinatorial generation. Algorithms to list all integers, tuples, permutations, combinations, partitions, set partitions, trees of a given size are also called combinatorial algorithms and are particular enumeration problems. Combinatorial algorithms are the subject of a whole volume of the Art of Computer Programming [72] and Knuth even confessed that these are its favorite algorithms; while I agree I would extend that appreciation to all enumeration algorithms.

Hundreds of different enumeration problems have now been studied, a partial list called "Enumeration of Enumeration Algorithms and Its Complexity" is maintained [101], and you are welcome to extend it! Some enumeration problems have important practical applications. Any database query is the enumeration of assignments of a formula, data mining relies on generating all frequent objects in a large database (frequent itemset [1], frequent subgraphs[64]), finding the minimal transversals of a hypergraph has applications in many fields [60] such as biology, machine learning, cryptography ...

A classical approach to enumeration is to see it as a variation on decision or search problem where one tries to get more information. As an example, consider the matchings of a graph, we may want to solve the following tasks.

- (decision problem) Decide whether there is a matching.
- (search problem) Produce a matching.
- (optimization problem) Produce a matching of largest cardinal.
- (counting problem) Count all matchings.
- (enumeration problem) List all matchings.

Usually, to analyze the complexity of a problem, we relate the time to produce the output to the size of the input. The originality of enumeration problem is that the output is usually very large with regard to the input. Hence, only measuring the *total time* to produce the whole set of solution is not informative since for most enumeration problems, including listing matchings, it is exponential.

To make enumeration complexity relevant, we must study finer complexity measures than just total time, since it does not allow to differentiate most enumeration problems. Moreover, we use three different parameters to characterize an instance: its size as in the classical setting, the size of the output (or its cardinal since it is a set of solutions) and the size of a single solution in the output.

The simplest way to improve on the analysis of enumeration algorithm is to evaluate how the *total time* to compute all solutions relate to *the size of the input and of the output*. Algorithms whose complexity is given in this fashion are often called *output sensitive*, by contrast to *input sensitive algorithms* [53]. Note that output sensitivity is relevant even when the number of objects to produce is small, in computational geometry it allows to give better complexity bound, for instance on the convex hull problem [26]. When the *total time* is polynomial in the size of the input and the output, the algorithm is said to be *output polynomial* (or sometimes total polynomial time). Output polynomial is a good measure of tractability when *all* elements of a set must be generated, for instance to count the number of solutions or to compute some statistics on the set of solutions.

Often, output polynomial is not restrictive enough given the outstanding number of solutions and we must ask for a total time *linear* in the number of solutions. In that case, the relevant complexity measure is the total time divided by the number of solutions called *amortized time* or *average delay*. Many enumeration algorithms generating combinatorial objects are in constant amortized time or CAT, the unrooted trees of a given size [105], the linear extensions of a partial order [89] or the integers given in Gray code order [73]. Uno also proposed in [97] a general method to obtain constant amortized time algorithms, which can be applied, for instance, to find the matchings or the spanning trees of a graph.

Enumeration algorithms are also often used to compute an optimal solution by generating all admissible solutions. For instance, finding maximum common subgraphs up to isomorphism, a very important problem in cheminformatics, is NP-hard and is solved by listing all maximal cliques [46]. The notion of best solution is not always clear and enumeration is then used to build libraries of interesting objects to be analyzed by experts, as it is done in biology, chemistry or network analytics [5, 10, 13]. In particular, when confronted to a multicriteria optimisation problem, a natural approach is to enumerate the Pareto's frontier [88, 98, 12]. In all these applications, if the set of solutions is too large, we are interested in generating the largest possible subset of solutions. Hence, a good enumeration algorithm should guarantee that it will find as many solutions as possible in a predictable amount of time. In this case, *polynomial incremental time* algorithms are more suitable: an algorithm is in polynomial incremental time if the time needed to enumerate the first k solutions is polynomial in k and in the size of the input. Such algorithms naturally appear when the enumeration task is of the following form: given a set of elements and a polynomial time function acting on tuples of elements, produce the closure of the set by the function. One can generate such closure by iteratively applying the function until no new element is found. As the set grows, finding new elements becomes harder. For instance, the best algorithm to generate all circuits of a matroid uses a closure property of the circuits [71] and is thus in polynomial incremental time. The fundamental problem of generating the minimal transversals of a hypergraph can also be solved in subexponential incremental time [54] and some of its restrictions in polynomial incremental time [48].

However, when one wants to process a set in a streaming fashion such as the answers of a database query, incremental polynomial time is not enough and we need a good *delay* between the output of two consecutive solutions, usually bounded by a polynomial in the input size. We refer to such algorithms as *polynomial delay* algorithms. Many problems admit such algorithms, e.g. enumeration of the cycles of a graph [91], the satisfying assignments of some tractable variants of SAT [36] or the spanning trees and connected induced subgraphs of a graph [6]. All polynomial delay algorithms are based on few methods such as *backtrack search* (also called flashlight search or binary partition) or *reverse search*, see [78] for a survey.

When the size of the input is much larger than the size of one solution, think generating subsets of vertices of a hypergraph or small queries over a large database, polynomial delay is an unsatisfactory measure of efficiency. The good notion of tractability is *strong polynomial delay*, i.e. the delay is polynomial *in the size of the last solution*. A folklore example is the enumeration of the paths in a DAG, which is in delay linear in the size of the last generated path. More complex problems can then be reduced to generating paths in a DAG, such as enumerating the minimal dominating sets in restricted classes of graphs [58].

Contrary to classical complexity classes, none of the classes introduced in this column but EnumP, the equivalent of NP, have complete problems. Because of that, there are no definite notion of reduction between enumeration problems, and for each proof that some enumeration problem is harder than another, there is a new reduction. To overcome the lack of completeness result, several works restrict themselves to smaller families of enumeration problem in the hope of better classifying their complexity: assignments of SAT formula [36], homomorphisms [19], subsets given by saturation operators [80], FO queries over various structures [93], maximal subgraphs [27, 31, 30]. . .

Organization In Section 2, enumeration problems and the related computation model are defined, with an emphasis on the consequences of several definitional choices. Then we introduce complexity classes related to three complexity measures, the total time in Section 3, the incremental time in Section 4 and the delay in Section 5. For all these classes, we provide separation and characterizations when possible. In Section 6, we describe the lack of hardness and lower bound results for enumeration problems and show some perspective to overcome this sad state of affairs. Finally in Section 7, we briefly sketch several ways to extend the complexity classes introduced (solution order, space, randomization, FPT. . .) and present several alternative approaches to the task of listing solutions exhaustively.

2 Enumeration problems

Let Σ be a finite alphabet and Σ^* be the set of finite words built on Σ. We denote by $|x|$ the length of $x \in \Sigma^*$. Let $A \subseteq \Sigma^* \times \Sigma^*$ be a binary predicate, we write $A(x)$ for the set of y such that $A(x, y)$ holds. The enumeration problem $\textsc{Enum}{\cdot}A$ is the function which associates $A(x)$ to x. The element x is often called the instance or the input, while an element of $A(x)$ is called a solution. We denote the cardinal of a set S by $|S|$.

In this column, we only consider predicates A such that $|A(x)|$ is finite for all x. This assumption could be lifted and the definitions on the complexity of enumeration adapted to the infinite case. This is not done here because infinite sets of solutions behave quite differently when studying the complexity of their enumeration. However, there are many natural infinite enumeration problems such as listing all primes or all words of a context-free language [52].

We can further reduce the set of enumeration problems by adding constraints on their solutions. First, the size of each solution can be bounded by a function of the instance. It is reasonable since in many problems the size of a solution is fixed, known beforehand and not too large, otherwise we would not even try to produce them. A predicate A is said to be *polynomially balanced* if for all $y \in A(x)$, $|y|$ is polynomial in $|x|$.

Let $\textsc{Check}{\cdot}A$ be the problem of deciding, given x and y, whether $y \in A(x)$. A constraint we could impose on enumeration problem is to be able to check efficiently whether a string is a solution. To do that, we ask for $\textsc{Check}{\cdot}A$ to be in polynomial time, a condition which holds for almost all practical enumeration problems.

From a polynomially balanced predicate A with $\textsc{Check}{\cdot}A$ in polynomial time, we can define an NP problem by asking whether $A(x)$ is empty or a $\#\mathsf{P}$ problem by asking for $|A(x)|$. It is then natural to define a class containing such enumeration problems, analogous to NP.

Definition 2.1. The class EnumP is the set of all problems $\textsc{Enum}{\cdot}A$ where A is polynomially balanced and $\textsc{Check}{\cdot}A \in \mathsf{P}$.

The problems in EnumP can be seen as the task of listing the solutions (or witnesses) of NP problems. One good property of EnumP is to be stable under several reductions and having some complete problem as explained in Section 6. However, it is hard to define a reduction which makes all hard problems of EnumP complete and this difficulty appears in the definition of the polynomial hierarchy for enumeration that we mention in Section 7.

Finally, remark that in the definition of EnumP no specificity of enumeration are taken into account. We are able to define it before even specifying the computation model or the complexity measures, as it only relies on the complexity of deciding the auxiliary problem CHECK·A.

Computation model The computational model is the random access machine model (RAM) with comparison, addition, subtraction and multiplication as its basic arithmetic operations and an operation `Output`(i, j) which outputs the concatenation of the values of registers $R_i, R_{i+1}, \dots, R_j$. RAM machine have been introduced by Cook and Reckhow [34, 2]; for variants designed for enumeration see [7, 94]. All instructions are in constant time except the arithmetic instructions which are in time logarithmic in the sum of the integers they are called on.

A RAM machine solves ENUM·A if, on every input $x \in \Sigma^*$, it produces a sequence $y_1, \dots, y_n$ such that $A(x) = \{y_1, \dots, y_n\}$ and for all $i \neq j$, $y_i \neq y_j$, that is *no solution must be repeated*! We may assume that all registers are initialized to zero, and the space used by the machine at some point of its computation is the sum of the length of the integers up to the last registers it has accessed. We define by $T_M(x, i)$ the time taken by the machine M on input x before the ith `Output` instruction is executed. Usually we drop the subscript M and write $T(x, i)$ when the machine is clear from the context. The delay of a RAM machine which outputs the sequence $\{y_1, \dots, y_n\}$ is the maximum over all $i \leq n$ of the time the machine uses between the generation of y_i and y_{i+1}, that is $\max_i T(x, i+1) - T(x, i)$. In some works, preprocessing and postprocessing times are considered separately from the delay. It is extremely rare to need more time for deciding if the enumeration is finished than to output a solution, hence we consider here that there is no postprocessing: enumeration stops as soon as the last solution is output. To make small complexity classes interesting, it is important to allow preprocessing, that is $T_M(x, 0)$, to be larger than the delay and we will mention it when appropriate.

Why a RAM instead of a Turing Machine ? While the RAM machine better maps to real computers and is thus better to measure precisely the complexity of algorithms, it can be simulated with cubic slowdown by a Turing Machine [34, 87]. A polynomial time Church's Thesis states that all realistic computational models are equivalent up to polynomial slowdown[1], which makes the computation model irrelevant for classical complexity. However, we can isolate a sequence of operations *during the execution* of a RAM which takes exponentially more time on a Turing Machine which simulates it. The RAM has the power of *indirection*: it can access any address in constant time, while the Turing machine should traverse all its tape to read some cell. This allows to use dictionary data structures essential

[1]the thesis is not true for quantum computers, but we still do not know whether they are physically implementable

in enumeration such as AVL trees or tries [35] which gives linear time access to objects inside an exponential set.

Why such a constant time OUTPUT instruction? The choice of the OUTPUT instruction and of its complexity is only relevant for algorithms with a sublinear delay, in particular a constant delay. In all definitions of RAM machine for enumeration [7, 94, 78, 17] the OUTPUT instruction can be issued in constant time. This is required to make constant delay interesting by capturing problems like Gray code enumeration or query answering, otherwise only a constant number of constant size solutions can be generated in constant delay. This is similar to the definition of logarithmic space, where machine can write an output in a special tape which is not taken into account in the space used [87]. Constant time output is meaningful, when only the deltas between solutions rather than solutions are output. It is also relevant, when instead of storing solutions we only need to do a constant time operation on each solution such as counting them or evaluating some measure which depends only on the constant amount of changes between two consecutive solutions.

The originality of the RAM model proposed in this column is to allow outputting solutions at different positions of the memory, to really take advantage of indirection. In previous models the size of the solution was implicit, here we make it explicit and also maintain its position in memory. This choice does not seem a big stretch from reality since, in practice, a programmer does not even precisely control where information is stored in memory. It allows to list with constant delay consecutive solutions which may differ by an unbounded number of elements, which is not possible in the traditional model with fixed registers for the output.

Why this cost model ? The cost model used to take into account the different operations of the RAM has no impact on complexity classes defined by a polynomial bound, so the choice is mostly arbitrary. We choose to count addition and multiplication as a linear number of operations in the size of their arguments. We could choose to count the addition as a unit time operation, but not the multiplication otherwise we can generate doubly exponential numbers in linear time. Note that we need to allow unbounded integers in registers to be able to deal with large data structures.

For small complexity classes, such as linear delay or constant delay, the choice of the cost model becomes extremely relevant. In such situations, sometimes implicitly, the *uniform cost model* (see [34, 2]) is chosen: addition, multiplication and comparison are in constant time. However, if the input is of size n, the machine has $\log(n)$ word-size, i.e. integers in registers are bounded by n. This model is robust enough to define linear time computation [59] and constant delay in enumeration [43, 7]. It is similar to word RAM model, a transdichotomous model [55], used to give finer and more realistic bounds for data structures. In some enumeration algorithms, all solutions must be stored and they can be 2^n of them. Hence, restricting integers in registers to be of size $\log(n)$ or even $k \log(n)$ for a fixed k is not sufficient to address all the memory needed to store the solutions. Hence, rather than bounding the register size, a good compromise is to take as cost of an instruction the logarithm of the sum of its arguments *divided by n*. Alternatively, some constant time operations on unbounded integers such as comparison and incrementation can be allowed.

3 Output polynomial time

To measure the complexity of an enumeration problem, we consider the total time taken to compute all solutions. Since the number of solutions can be exponential with regard to the *input*, it is more relevant to give the total time as a function of the size of the input and of *the output*. In particular, we would like it to be polynomial in the number of solutions; algorithms with this complexity are said to be in output polynomial time or sometimes in polynomial total time.

Definition 3.1 (Output polynomial time)**.** A problem $\textsc{Enum}\cdot A \in$ EnumP is in OutputP if there is a polynomial $p(x, y)$ and a machine M which solves $\textsc{Enum}\cdot A$ and such that for all x, $T(x, |A(x)|) < p(|x|, |A(x)|)$.

For instance, if we see a polynomial as a set of monomials, then classical algorithms for interpolating multivariate polynomials from their values are output polynomial [107] as they produce the polynomial in a time proportional to the number of its monomials.

The classes EnumP and OutputP may be seen as analogue of NP and P for the enumeration. It turns out that their separation is equivalent to the $\mathsf{P} = \mathsf{NP}$ question, since an algorithm in OutputP allows to decide whether there is at least one solution in polynomial time.

Proposition 3.1 (Folklore, see [21])**.** OutputP = EnumP *if and only if* $\mathsf{P} = \mathsf{NP}$.

4 Incremental polynomial time

When generating all solutions is already too long, we want to be able to generate at least some. Hence, we should measure (and bound) the total time used to produce a given number of solutions. This dynamic version of the total time is called *incremental time*. Given an enumeration problem A, we say that a machine M

solves Enum·A in incremental time $f(m)g(n)$ if on every input x, M enumerates m elements of $A(x)$ in time $f(m)g(|x|)$ for every $m \leq |A(x)|$.

Definition 4.1 (Incremental polynomial time)**.** A problem Enum·$A \in$ EnumP is in IncP$_a$ if there is a machine M which solves it in incremental time $O(m^a n^b)$ for b constant. Moreover, we define IncP $= \bigcup_{a \geq 1}$ IncP$_a$.

In the definition of IncP$_a$, we usually allow preprocessing polynomial in the input to make the class more robust. Remark that allowing arbitrary polynomial preprocessing does not modify the class IncP, since this preprocessing can always be seen as the polynomial time before outputting the first solution.

Let A be a binary predicate, AnotherSol$_\mathsf{A}$ is the search problem defined as, given x and a set $\mathcal{S}$, find $y \in A(x) \setminus \mathcal{S}$ or answer that $\mathcal{S} \supseteq A(x)$ (see:[94, 37]). The problems in IncP are the ones with a polynomial search problem, as stated in the following proposition.

Proposition 4.1 (Folklore)**.** *Let A be a predicate such that* Enum·$A \in$ EnumP. AnotherSol$_\mathsf{A}$ *is in* FP *if and only if* Enum·A *is in* IncP.

The class IncP is usually defined as the class of problems solvable by an algorithm with a delay polynomial in the number of already generated solutions and the size of the input.

Definition 4.2 (Usual definition of incremental polynomial time.)**.** A problem Enum·$A \in$ EnumP is in UsualIncP$_a$ if there is a machine M which solves it such that for all x and for all $0 < t \leq |A(x)|$, $|T(x,t) - T(x,t-1)| < ct^a|x|^b$ for b and c constants.

This alternative definition is motivated by saturation algorithms, which produce solutions by applying some polynomial time rules to enrich a set of solutions until saturation. There are many saturation algorithms, for instance to enumerate circuits of matroids [71] or to compute closure by set operations [79].

With our definition, we better capture the fact that investing more time guarantees more solutions to be output, which is a bit more general at first sight than bounding the delay because the time between two solutions is not necessarily regular. It turns out that these classes are the same by *amortization*, that is storing the output solutions in a buffer and outputting them regularly from the buffer.

Proposition 4.2 (Folklore, see [20])**.** *For every $a \in \mathbb{N}$,* IncP$_{a+1}$ = UsualIncP$_a$.

The amortization method plus a data structure to detect duplicates solutions such as a trie, allow to deal with bounded repetition of solutions and is often used to simplify the design of algorithms, see for instance the well-named Cheater's Lemma [22].

Since IncP is characterized by the search problem $\mathsf{AnotherSol_A}$, it is related to the class TFNP introduced in [82]. A problem in TFNP is a polynomially balanced polynomial time predicate A such that for all x, $A(x)$ is not empty. An algorithm solving a problem A of TFNP on input x outputs one element of $A(x)$. The class TFNP can also be seen as the functional version of $\mathsf{NP} \cap \mathsf{coNP}$. It turns out that the separation of IncP and OutputP is equivalent to the separation of TFNP and FP. The proof is based on modifying $\mathsf{AnotherSol_A}$ into a problem of TFNP, so that, assuming $\mathsf{TFNP} = \mathsf{FP}$, a solution to this problem is found which can then be used to solve $\mathsf{AnotherSol_A}$.

Proposition 4.3 ([20])**.** $\mathsf{TFNP} = \mathsf{FP}$ *if and only if* $\mathrm{IncP} = \mathrm{OutputP}$.

The class OutputP is conditionally different from EnumP as shown in Proposition 3.1 and it is also strictly larger than IncP by Proposition 4.3. However, no natural problem is known to be inside OutputP but not in IncP.

Open problem 1. *Find a natural problem in* OutputP *not in* IncP. *It should be a problem for which* AnotherSol *is hard, but not its decision version, where the existence of another solution is asked.*

We can get a finer separation of classes than in Proposition 4.3: $(\mathrm{IncP}_a)_{a\in\mathbb{R}^+}$ form a strict hierarchy inside IncP. We need to assume some complexity hypothesis since $\mathsf{P} = \mathsf{NP}$ implies $\mathrm{IncP} = \mathrm{IncP}_1$. Since we need to distinguish between different polynomial complexities as in fine grained complexity, we also rely on the Exponential Time Hypothesis (ETH). It states that there exists $\epsilon > 0$ such that there is no algorithm for 3-SAT in time $\tilde{O}(2^{\epsilon n})$ where n is the number of variables of the formula and $\tilde{O}$ means that we have a factor of $n^{O(1)}$.

Theorem 4.1 ([20])**.** *If* ETH *holds, then* $\mathrm{IncP}_a \subsetneq \mathrm{IncP}_b$ *for all* $a < b$.

The theorem is proved by considering a modified version of SAT with a carefully chosen number of trivial solutions and an exponential repetition of the real ones. Then, if $\mathrm{IncP}_a = \mathrm{IncP}_b$ we can find a solution of any SAT formula slightly faster than brute force by using the incremental algorithm, which in turn proves a larger collapse of the IncP_a classes. Applying this trick enough times proves the theorem. Using a similar proof, the same strict hierarchy for OutputP can be obtained.

5 Delay

In this section, we study complexity classes derived from a constraint on the delay, depending either on the input or the size of a single solution.

5.1 Polynomial delay

Definition 5.1 (Polynomial delay)**.** A problem ENUM·$A \in$ EnumP is in DelayP if there is a machine M which solves it such that for all x and for all $0 < t \leq |A(x)|$, $|T(x,t) - T(x,t-1)| < C|x|^a$ for constants C and a.

Observe that, by definition, DelayP = UsualIncP$_0$ and is thus included in IncP by Proposition 4.2. Polynomial delay is the most usual notion of tractability in enumeration, both because it is one of the best possible guarantees (regularity and linear total time) and paradoxically because it is relatively easy to obtain. Indeed, most methods used to design enumeration algorithms such as backtrack search with an easy extension problem [80], or efficient traversal of a supergraph of solutions [75, 6], yield polynomial delay algorithms when they are applicable.

Contrarily to IncP, there is no characterization of DelayP related to the complexity of some search or decision problem, but there are some interesting implications. Let EXTSOL·A be the problem of deciding, given x and w_1, whether there is $s = w_1 w_2$ such that $s \in A(x)$. The set $A(x)$ can be divided into solutions beginning by 0 and 1 and so on recursively and EXTSOL·A can be used to decide whether those subsets of solutions are empty. This method is an efficient backtrack search, often called binary partition or flashlight search (see e.g. [80]).

Proposition 5.1. *If* EXTSOL·$A \in \mathsf{P}$*, then* ENUM·$A \in$ DelayP.

Consider any graph property $\mathcal{P}$ closed by induced subgraph. Given a graph, the problem is to generate all its maximal induced subgraphs satisfying $\mathcal{P}$, for instance generating the maximal cliques of a graph if the property is to be a complete graph. The *input restricted problem* is also the problem of generating maximal induced subgraphs, with the additional constraint that there is a vertex in the input graph G such that $G \setminus v \in \mathcal{P}$. The general problem is in DelayP if the input restricted version is in polynomial time (and thus has only a polynomial number of solutions). This method helps design algorithms, and equivalences with the restricted problem can be obtained for IncP and OutputP, see [27]. This is a generalization of a method by Lawler et al. [75] on families of sets closed by subsets, where the enumeration is related to the so called $I \cup j$ problem.

Since an enumeration problem asks for a set of solutions, we can build new problems by acting on this set: the predicate $A \cup B$ is defined as $(A \cup B)(x) = A(x) \cup B(x)$ and we get a new problem ENUM·$(A \cup B)$. This definition can easily be adapted to any set operation such as the complement or the intersection. We say that a complexity class C is *stable* under some operation, say the union, when ENUM·A, ENUM·$B \in C$ implies ENUM·$A \cup B \in C$. It turns out that DelayP is stable for several operations.

Theorem 5.1 ([94])**.** *The classes* OutputP*,* IncP*,* DelayP *are stable under union, Cartesian product and polynomial size deletion.*

As a consequence, these operations can be used to design reductions, as shown in Section 6 or to simplify the design of good algorithms by focusing on simpler problems. To obtain stability under union for smaller classes, like constant delay, we need additional hypotheses, either the union is disjoint or the $\textsc{Check}{\cdot}A$ problem is in constant time. Several other operations do not leave any enumeration classes stable but EnumP because they allow to build an EnumP-complete problem from simpler ones.

Theorem 5.2 ([94])**.** *If* $\mathsf{P} \neq \mathsf{NP}$ *the* DelayP, IncP, OutputP *are not stable under unbounded deletion, intersection or complement.*

5.2 Strong polynomial delay

When the input is huge with regard to a single solutions, we would like a sublinear delay, or even a delay which depends only on the size of the generated solutions. In the following definition, a preprocessing polynomial in the size of the input is allowed, since the input should at least be read before outputting solutions.

Definition 5.2 (Strong Polynomial delay)**.** A problem $\textsc{Enum}{\cdot}A \in$ EnumP is in SDelayP if there is a machine M which enumerates $A(x) = \{y_1, \dots, y_n\}$ for all x and for all $0 < t \leq n$, $|T(x,t) - T(x,t-1)| < C|y_t|^a$ for constants C and a.

Presently, very few problems have strong polynomial delay algorithms: generating the assignments of $\exists FO$ formulas with second order free variables [44] or existential MSO formulas over bounded tree width structures [3]. However, even extremely simple problems such as computing the closure by union of a set system [80], have a linear delay algorithm but do not seem to have a strong polynomial delay algorithm.

Open problem 2. *Prove that* SDelayP $\subsetneq$ DelayP *modulo some decision complexity hypothesis.*

Studying EnumDNF, the enumeration of assignments of a DNF formula, could help prove the separation of SDelayP and DelayP. The structure of the assignments of a DNF is very simple: it is the *union* of the assignments of its terms. The main difficulty to obtain a strong polynomial delay algorithm for EnumDNF is that *the union is not disjoint* and just enumerating the solutions of each term causes repetitions of solutions. Solution repetitions because of non disjoint union is a common problem in enumeration and this issue appears in its simplest form when solving EnumDNF. We hope that understanding finely the complexity of this problem and giving better algorithm to solve it will shed some light on more general problems, that is why we propose the following conjectures.

Conjecture 5.1 (DNF Enumeration Conjecture)**.** EnumDNF $\notin$ SDelayP.

We can even state a stronger variant, similar in precision to ETH.

Conjecture 5.2 (Strong DNF Enumeration Conjecture)**.** There is no algorithm solving the problem EnumDNF in delay $o(m)$ where m is the number of terms of the DNF.

In [20], several stronger forms of these conjectures on restricted formulas such as monotone DNF and k-DNF, or adapted to the average delay are refuted.

Open problem 3. *Prove or disprove the (strong) DNF Enumeration Conjecture modulo some decision complexity hypothesis.*

5.3 Four flavors of constant delay

The class of problems solvable with a constant delay is very sensitive to the way the computational model and the complexity measure are defined. There are at least four slightly different notions of constant delay used by different communities. In all cases, we assume that operations on integers less than n are in constant time, with n the size of the input.

Constant delay In several RAM models, the output registers are fixed and an output solution is always contained in the first of these registers. Hence, two consecutive solutions must only differ by a fixed number of elements, such an ordering of solutions is often called a Gray code. Moreover; it should be possible to go from a solution to the next in constant time, such an algorithm is sometimes called loopless in the context of Gray code. There are many such algorithms for the generation of combinatorial objects [100, 92, 72]: the integers less than 2^n, combinations, Dyck words, . . .

Constant delay with dynamic amortization The use of the generalized OUTPUT instruction presented in this column allows to amortize a slow enumeration process with a fast one. It can be used to generate assignments of a k-DNF in constant time [20]. Constant delay can be obtained even when the solutions do not follow a Gray code order.

Constant amortized time An algorithm has a constant amortized time (CAT) if its total time divided by the number of solutions is constant, i.e. its average delay is constant. Consider the generation of the integers between 0 and $2^n - 1$ in the usual order. Adding one to an integer number may change up to n bits, but on average only two bits are changed and the time needed is proportional to the number of changed bits. Hence, on average over the whole enumeration, only a constant number of operations per solution is required. See Ruskey's book [92] for many examples of CAT algorithms. Uno's push-out method [97] also helps turn backtracking algorithms into CAT algorithms.

Remark that for some problems, such as the enumeration of unrooted trees [105], it has been later proved that they admit a proper constant delay algorithm [84]. Amortization may be used to turn a CAT algorithm to constant delay algorithm if the generation of solutions is regular enough. Sometimes, the solutions to be generated are organized as nodes of a tree and given by some traversal of the tree. Using Feder's trick, introduced for 2CNF [51], of generating solutions of even depth when going down the tree and solutions of odd depth when going up, one may turn a CAT algorithm into a constant delay one without relying on amortization.

Open problem 4. *Give a sufficient condition to make constant amortized time algorithms constant delay, possibly using exponential space. Can we always do that for algorithms obtained by push-out amortization?*

FPT delay When doing query evaluation, the query is often considered fixed, and we say that we consider the *data complexity*. Hence, the number of free variables of the query is also fixed, let us denote it by k. A solution is a k-tuple of elements from the structure. Let n be the size of the structure, then k integers of size $\log(n)$ bits are necessary to encode a solution. Since we have assumed a uniform cost measure, these elements can be dealt with in constant time. Note that there are at most n^k solutions, hence to make the class interesting, the preprocessing should be bounded by something stricter than any polynomial. The choice is often to consider a linear preprocessing.

Durand and Granjean have shown that FO queries over bounded degree structures can be done with constant delay and linear preprocessing [43]. Following that result, many query languages have been proven to have a constant delay algorithm, see the survey of Segoufin [93]. Several enumeration algorithms for solving FO queries over sparse models (see [99]) can be executed by a jumping automaton on graph or JAG [33], which represents an input by a colored graph and is only allowed to move pebbles along edges as a computation step. A JAG with a fixed number of pebbles is more restricted than a constant delay algorithm. One could hope to prove lower bounds for enumeration using JAGs, in fact they have been introduced to prove space lower bounds for problems like connectivity [102].

Open problem 5. *[Suggested by Segoufin] Can we prove unconditionally that there is some simple FO query, say $\exists z(R(x,z) \wedge S(z,y))$, which cannot be solved using a JAG with a constant number of pebbles and linear preprocessing?*

6 Reduction and lower bounds

In this section, we study the problem of proving that an enumeration problem is hard for some classes. We first show that many different reductions are used to prove that some problem is harder than another. Unfortunately, we have no satisfying notion of completeness as in classical complexity, which makes relative hardness of problems less satisfying. Then we show a few methods to prove that an enumeration is not in some class relying on decision problems. Finally, in the spirit of fine grained complexity [104], we propose to base hardness of enumeration on the supposed hardness of well-studied enumeration problems such as the enumeration of minimal transversal in hypergraphs.

6.1 Reductions

In this section we briefly review different notions of reductions for enumeration problems. One required property is that complexity classes introduced in the previous sections are closed under reduction. It means that if we can reduce Enum·A to Enum·B and that Enum·$B \in C$ for some class C, then Enum·$A \in C$. This property guarantees that reduction can serve to provide algorithms and not just hardness. All considered reductions are also transitive. The parsimonious reduction for counting problems enforces equality of numbers of solutions. We can adapt it by providing an explicit way to realize the bijection between solutions.

Definition 6.1 (Parsimonious Reduction)**.** Let Enum·A and Enum·B be two enumeration problems. A parsimonious reduction from Enum·A to Enum·B is a pair of polynomial time computable functions f, g such that for all x, $g(x)$ is a bijection between $A(x)$ and $B(f(x))$.

An EnumP-complete problem is defined as a problem in EnumP to which any problem in EnumP reduces. The problem Enum·$3SAT$, the task of listing all solutions of a 3-CNF formula is EnumP-complete, since the reduction used in the proof that SAT is NP-complete [32] is parsimonious. The parsimonious reduction is enough to obtain EnumP-complete problem, but is usually too restrictive to make some hard problems complete. Let us consider the predicate $SAT_0(\phi, x)$ which is true if and only if x is a satisfying assignment of the propositional formula ϕ or x is the all zero assignment, then $SAT_0(\phi)$ is never empty and therefore many problems of EnumP cannot be reduced to Enum·SAT_0 by parsimonious reduction, while, from an enumeration perspective, it is exactly as hard as Enum·$3SAT$.

To overcome this problem, we can generalize the parsimonious reduction as much as possible, so that DelayP is stable under the designed reduction. To make Enum·SAT_0 complete, it is enough to allow in parsimonious reduction the addition or deletion of a polynomial number of solutions themselves produced in polynomial time. DelayP is stable under such reduction because of Proposition 5.1. Several reductions for OutputP, IncP and DelayP have been given by Mary in [78]. We give here a polynomial delay reduction, which associates to each solution of some problem a subset of solutions of another problem such that all these subsets partition the desired set of solutions.

Definition 6.2 (Polynomial delay reduction from [78])**.** Let $\textsc{Enum}\cdot A, \textsc{Enum}\cdot B \in$ EnumP and f, g two polynomial time computable functions. The pair (f, g) is a polynomial delay reduction if there is a polynomial p such that:

- $\bigcup_{s \in B(f(x))} g(s) = A(x)$ (all solutions of A are obtained)
- $|\{s \in B(f(x)) \mid g(s) = \emptyset\}| \leq p(|x|)$ (all but a polynomial number of solutions of B give solutions of A)
- $\forall s_1,\ s_2 \in B(f(x)),\ g(s_1) \cap g(s_2) = \emptyset$ (no repetition of solutions)

Using this reduction, it has been proven that the problem of listing minimal transversals of a hypergraph is equivalent to listing the minimal dominating sets of a graph [67], a result which has motivated the study of enumeration of minimal dominating sets over various graph classes [68, 14].

We can relax polynomial delay reductions by allowing a polynomially bounded number of repetitions, such reductions are called e-reduction in [37]. Instead of mapping a solution of B to a polynomial number of solutions of A, we could even allow a polynomial delay algorithm to generate any number of solutions of A from a solution of B. We can further be generalized by defining Turing reduction for enumeration, that is allowing access to an oracle solving an enumeration problem using two additional instructions, one to begin an enumeration, the other to get the next solution. When the oracle machine computing the reduction has a polynomial delay or an incremental polynomial time, the reductions are called D or I reductions [37].

Finally, the notion of polynomial delay reduction may be refined when considering hardness results within DelayP as in fine-grained complexity, in particular hardness for strong polynomial delay and constant delay. It can be done by requiring that, for a reduction (f, g), $g(s)$ can be enumerated with strong polynomial delay or constant delay. Such a reduction is used to prove that several saturation problems are not in SDelayP if listing the models of monotone DNF [80] is not in SDelayP.

No complete problems are known for the complexity classes we have introduced but EnumP with respect to any reduction. In fact, artificial constructions using padding, as presented for SAT and parsimonious reduction, are often enough to build a hard but not complete problem. This emphasises the need to find other methods to prove lower bounds for specific problems.

6.2 Known lower bounds

Relation to decision To any enumeration problem ENUM·A, we can associate EXIST·A the problem of deciding whether $A(x) = \emptyset$. If P $\neq$ NP, any problem such that EXIST·A is NP-hard cannot be in OutputP. Hence, any variation on the enumeration of witnesses of NP-hard problems is not in OutputP.

Recall that, by Proposition 4.1, a problem ENUM·$A \in$ IncP if and only if $\mathsf{AnotherSol}_\mathsf{A} \in$ FP. Hence, if we assume P $\neq$ NP and that $\mathsf{AnotherSol}_\mathsf{A}$ is NP-hard, then ENUM·$A \notin$ IncP. This has been used to prove that several problems are not in IncP: enumerating maximal models of Horn formula [69], enumerating vertices of a polyhedron [70] and dualization in lattice given by implicational bases [41].

Barrier to a natural method The most used method to obtain a polynomial delay algorithm is showing that EXTSOL·A is polynomial. Hence showing that it is in fact NP-complete may rule out the use of backtrack search. Extension problems have been studied for their own sake and there are many NP-hardness results, see [24, 23] and the references therein. In particular, the extension problem of the minimal transversals is NP-hard [16].

Unconditional lower bounds Unconditional lower bounds are extremely hard to obtain: after fifty years of research, is not even known whether SAT requires superlinear time. To obtain unconditional lower bounds in enumeration, the easiest approach is to show that a problem is not in a very small complexity class, like constant delay with polynomial preprocessing, and even to restrict the computational model. That is why we have proposed the Open Problem 5 on the JAG computational model. Indeed, the best known separations from classical complexity are for classes of problems solvable by family of boolean circuits: strict hierarchy inside AC^0 [62], polynomial size monotone circuits cannot decide the existence of a k-clique [90] and ACC $\neq$ NEXP [103].

Open problem 6. *Can we define meaningful circuit classes for enumeration and prove unconditional separations of these classes? Can we relate circuit classes for enumeration to parallel computation or descriptive complexity?*

Lower bound from fine grained complexity Instead of assuming P $\neq$ NP, we may need to make stronger assumptions. If we assume that $3SUM$ cannot be solved faster than $O(n^2)$, there are several lower bounds on the complexity of generating all triangles of a graph [74]. Some lower bounds involving the number of triangles can be rephrased as stating that the delay is at least $O(m^{1/3})$, with m the number of edges. Another example is the characterization of the complexity of an acyclic FO query: either it is in some class called CCQ and its assignments can be enumerated in constant delay or it is not in CCQ and its assignments cannot be enumerated with constant delay, unless the product of boolean matrices can be computed in $O(n^2)$ [8]. Finally, an algorithm that generates maximal solutions of any strongly accessible set system has a worst case time complexity of $\Omega(t2^{q/2})$, unless SETH is false [30]. Here t denotes the number of solutions and q their maximal size.

Open problem 7. *Assuming* SETH*, prove a weak lower bound: there is no* constant delay *or* strong polynomial delay *algorithm for generating the minimal transversals of a hypergraph, the circuits of a binary matroid or the maximal cliques of a graph.*

Reduction to well studied problem Rather than assuming complexity hypotheses on decision problems, as in the previous paragraph, we could base the hardness of an enumeration problem on an hypothesis on the hardness of another *enumeration* problem. The best example of this approach are the many problems which have been proven to be as hard or equivalent to enumerating the minimal transversals of an hypergraph, a problem which is still not proved to be in OutputP after 30 years of research. Among these problems are the dualization of a monotone boolean formula, the generation of vertices of an integral polytope [15], the enumeration of minimal keys, see [47, 60] for many examples.

We can use the conjectures made on the complexity of EnumDNF in Section 5.2 as a source of hardness. More precisely, if EnumDNF can be reduced to some problem Enum·A, then it should be a sign that Enum·A is not in SDelayP. If the enumeration of assignments of a *monotone* DNF formula (equivalently the ideals of some boolean poset) can be reduced to Enum·A then it is a sign that there is no algorithm with strong polynomial delay and polynomial space.

Open problem 8. *Propose your own hypothesis on some enumeration problem you have not been able to crack. In particular, is there a good problem that is in* IncP *but not believed to be in* DelayP *which can be used for reduction ?*

Open problem 9. *Could we transfer hardness between classes: prove that some problem, for instance* EnumDNF*, is not in* SDelayP *if and only if some other problem, for instance generating minimal transversals, is not in* IncP.

7 Alternative approaches

7.1 Variations on enumeration complexity

The definition of enumeration problems and complexity classes we gave can easily be modified and extended, yielding other complexity classes and separations. Some of these variations are mentioned here:

Checkability In the definition of the different enumeration classes, we could remove the hypothesis that the solutions are of polynomial size and certifiable in polynomial time. In that case, all class separations are unconditional, using the time hierarchy Theorem [61], see [21]. However, several properties do not hold, such as the closure of DelayP under union and the equivalences between separation of enumeration complexity classes and $\mathsf{P} \neq \mathsf{NP}$ or $\mathsf{FP} \neq \mathsf{TFNP}$.

Solution order Enumeration problems can be further constrained by requiring the solutions to be produced in a given order. When partial orders are allowed, it is a strict generalization of enumeration problems as defined here. From a practical point of view, it is a way to deal with a large number of solutions by requesting the enumeration of the best ones first and it relates to the problem of finding the k best solutions (see Section 7.2). From a structural complexity perspective, on one hand it has no effect on the class OutputP since the elements can be sorted after having been produced. On the other hand, it has a dramatic effect on DelayP: maximal independent sets can be enumerated in lexicographic order with polynomial delay but not in reverse lexicographic order [65]! However, hardness results with a fixed order are directly derived from the hardness of finding some largest solution and do not tell much about enumeration. Indeed, even trivial enumeration problems become hard when paired with the appropriate order (see Section 2.4 of [94]).

FPT enumeration Fixed parameter tractable classes can be easily derived from the enumeration classes presented here in the obvious manner, and kernelization can be extended to enumeration. Enumeration of vertex covers or weighted assignments of satisfiability problems admit enumeration algorithms with an FPT delay [39, 38].

Randomized enumeration Randomization can be added to enumeration algorithms, by allowing to produce a wrong set of solutions with a small probability. The only example of such an algorithm is for enumerating monomials of a polynomial [95], and randomization is required because polynomial identity testing needs to be solved as a subroutine. Interestingly, while randomization helps in many settings, it does not seem yet well exploited in enumeration.

Dynamic enumeration Recently, many enumeration algorithms from databases have been adapted to allow for updates of the input. It means that after modifying the input, e.g. adding or removing an edge of a graph, or changing the color of a node, the enumeration can be restarted without doing another preprocessing (or a very short one). For instance, enumerating the satisfying assignments of MSO formulas over trees can be done with a delay linear in each solution and logarithmic update [4].

Space complexity Only time restrictions have been considered in this column, but space has a huge role in practical enumeration algorithms and is often a bottleneck, for instance when generating maximal cliques [29]. We could define an equivalent of PSPACE by allowing a space polynomial in the input, without taking into account the output solutions. It is also relevant to ask for polynomial space and polynomial delay algorithms as a tractability measure. In fact, several algorithmic methods have been designed to traverse a supergraph of solutions with polynomial delay, while using only polynomial space [75, 6, 30].

When trying to prove lower bounds, it could be helpful to add the constraint of polynomial space. For instance, we could try to prove that the minimal transversals of a hypergraph cannot be computed with polynomial delay *and polynomial space*. Moreover, amortization which allows to trade space for regularity as in Proposition 4.2 cannot work when requiring polynomial space. Hence, with polynomial space IncP_0 may not be equal to DelayP. Some partial results [21] show that algorithms in incremental linear time with enough regularity are in polynomial delay using only an additional polynomial space but the general question is open.

Open problem 10. *Prove or disprove that* $\mathrm{IncP}_0 = \mathrm{DelayP}$ *with polynomial space.*

7.2 Alternative approaches

There are many alternative ways to consider the problem of listing solutions. In this column, we have studied the dynamic of algorithms solving exhaustive enumeration problems through incremental time or delay. It seems to be very hard to prove lower bounds or completeness results with these dynamic measures and approaches relying on classical notions of time complexity are more amenable to hardness results. From a practical point of view, due to the size of the solution space, non exhaustive approaches seem particularly relevant.

Input sensitive Input sensitive algorithms have their complexity expressed in terms of the input size only. The size of the solution space is thus a lower bound to their complexity. Since this size is typically exponential, their complexity is also exponential. In fact, these algorithms are often called exact exponential algorithms (see the book of Fomin and Kratsch [53]). The aim is to find algorithms, whose complexity is as close as possible to the maximal number of solutions. As a byproduct, analysis of input sensitive algorithms often provide upper bounds on the number of solutions. A typical example is the Bron–Kerbosch algorithm [18] to list all maximal cliques of a graph: there are at most $3^{n/3}$ maximal cliques in a graph with n vertices [83] and its complexity is in $O(3^{n/3})$. When there are less solutions than the maximal number, input sensitive algorithms should be inefficient, but they are sometimes efficient in practice [25].

Polynomial hierarchy In this column, we have presented EnumP and classes inside it, trying to characterize different notions of tractability for enumeration. To classify hard problems, for which finding a solution is already NP-hard, we need something similar to the polynomial hierarchy. Such classes have been built [37] by adding oracles in the polynomial hierarchy to polynomial delay or incremental polynomial time machine. Note that it is difficult to find a reduction notion for these classes which leaves them stable and yet allows for complete problems.

K-best solutions One way to deal with a large solution space is to reduce it arbitrarily. We can fix a parameter k and ask to enumerate only k solutions. We could also have some order on the solutions and ask for the k best ones. When k is not fixed but given as input the problem is very similar to the (ordered) enumeration of solutions with respect to incremental time. The k-shortest path and the k-minimum spanning tree problems are typical k-best problems with many applications as explained in a survey by Eppstein [49]. In this survey, some methods for generating k-best solutions are presented, and they are variations on methods also used for enumeration: binary partition or the $I \cup \{j\}$ method [75].

Jth solution To avoid to deal with the time and space needed to generate the whole solution set, it would be good to generate solutions on demand. This problem is sometimes called the jth solution problem, there is some order on the set of solutions and given an integer j, we are asked to generate the jth. Remark that if the jth solution problem is in polynomial time, the counting problem is also in polynomial time. Hence, only very simple problems admit a polynomial time algorithm for the jth solution, for instance listing the solutions of an FO query over a bounded degree structure [9].

Uniform generation A more relaxed variant of the jth solution problem is to ask for a random solution given uniformly at random, a problem deeply connected to approximate counting [63]. A polynomial time uniform generator can be turned into an exhaustive enumeration algorithm working with high probability by repeatedly generating solutions. The only difficulty is to know when to stop because there is no new solution anymore, but a bound can be derived from the Coupon Collector Theorem [50].

Theorem 7.1 ([57, 21]). *If* Enum·$A \in$ EnumP *has a polytime uniform generator, then* Enum·A *is in randomized* DelayP.

The algorithm used in Theorem 7.1 needs an exponential space for storing the solutions and some amortization. In general, we cannot get rid of the space, since Goldberg has given a tight lower bound on the product of space and delay [57]. However, when repetitions are allowed, we can build an exhaustive algorithm using polynomial space [21], keeping the good space use of random generation.

Open problem 11. *There are several methods to obtain a uniform generator: random walk on a reconfiguration graph [85] or Boltzmann samplers [42]. Can we get an efficient enumeration algorithm directly from these methods without building the random generator?*

How much does a free solution help? Often, it is NP-hard to even find a single solution or a few of them. But what if we are provided a solution to some NP hard problem, can we find a second one easily ? It turns out that it can be hard, for instance finding a second Hamiltonian circuit given one is NP-hard [87]. In fact, when looking for Nash equilibria in a two players game, there is always one and it is relatively easy to find (in PPAD [86]), while finding a second one is NP-hard [56].

Finding a second solution is a restricted version of the AnotherSol problem and it has been systematically studied by several authors [106, 66]. For some problem, deciding whether there is a second solution given one is easy: There are always an even number of Hamiltonian circuits in cubic graph, hence if one is given there must be a second. It is easy to build an artificial enumeration problem, for which giving a single solution makes the enumeration easy or for which it does not help.

Open problem 12. *Is there a natural* EnumP*-complete problem which can be solved in output polynomial time when one solution is given along with the input?*

7.3 New horizons

Enumeration problems are especially exposed to the problem of combinatorial explosion because even a constant delay algorithm cannot escape the cost of going through each solution, which can already be too large. Moreover, even if a modern computer equipped with a good algorithm can easily generate thousands of billions of spanning trees or shortest paths, no user will ever be able to make sense of such an amount of data. Hence, we should find ways to generate fewer solutions while still capturing the essence of the set of solutions. Following the ideas of several enumeration specialists, in particular Uno, we propose two promising and mostly unexplored directions.

Succinct representation It is often natural to represent a set of solutions by Cartesian product and union of smaller sets: the maximal cliques of a graph are the union of the maximal cliques of its connected components and the spanning forests of a graph are the Cartesian product of the spanning forests of its connected components. A constant delay algorithms to list spanning trees using a more involved Cartesian product decomposition is given in [28] as well as several examples of Cartesian decomposition for enumeration.

Usually, when confronted to an enumeration problem, one first tries to find a representation of the set of solutions which minimizes its size while allowing efficient computation on it like finding its size, its maximum, some statistic on its elements . . . It should be some middle ground between the set of solutions, which is easy to exploit but often too large to compute and store, and the instance, which represents the set of solutions very succinctly but is not directly useful for computation.

This approach is similar to knowledge compilation [40], where some boolean function is represented by a succinct logical circuit (an OBDD or a DNNF). While the cost of computing the representation may be large, after compilation all queries over the original function can be evaluated efficiently over the representation. In a recent result, Amarilli et al. [3] give a strong polynomial delay algorithm to enumerate the models of d-DNNF circuits used in knowledge compilation. They relate it to set circuits, which can be seen as way of generating a set of solutions by using only Cartesian products and *disjoint* unions of ground solutions. Several known enumeration problems such as the enumeration of the models of an MSO formula on a structure of bounded tree-width can then be reduced to the enumeration of the solutions of such circuits.

Open problem 13. *What enumeration problem can we capture using circuits of Cartesian products and* general *unions? Is it possible to generalize Cartesian product by some form of join without losing tractability?*

Open problem 14. *What decomposition of solution sets can be useful in enumeration outside of union and Cartesian product?*

Approximate representation The previous point was about compressing the set of solutions without loss. But we could also compress it with loss, that is producing a small representation which cannot be used to recover all solutions but only to approximate them.

This idea has been used to replace the Pareto's frontier of a problem by a much smaller number of approximate Pareto optimal points [88]. The lossy compression approach is also reminiscent of sketches used to obtain a succinct representation of a very large stream, while maintaining some property with high probability. For instance, using a logarithmic space, the number of distinct elements in a stream can be evaluated [45], or with a sublinear space the minimum spanning tree or maximal matching can be approximated [81].

To approximate a set of solutions $A(x)$, we can use some subset S such that $|S|$ is within some factor of $|A(x)|$. We could also allow non solutions to be in S and then both $|S \cap A(x)|$ and $|S \vartriangle A(x)|$ must be within some factor of $|A(x)|$. This kind of approximation based on the number of solutions is not very relevant: what if we get a large family of similar solutions but miss all the diversity of the solution set? To avoid this problem, an equivalence relation can be used to define similar solutions. Then, only one representative per equivalence class should be enumerated. Representatives are often harder to enumerate, a good example is the enumeration of classes of trees or graphs *up to isomorphism*, see Chapter 8 of [92].

More generally, we could equip the solutions with some distance and try to cover them all by as few of them as possible. Given an instance x of a problem $\textsc{Enum}{\cdot}A$ and an integer d, we define a d-cover of $A(x)$ as $S \subseteq A(x)$ such that for all solutions of $A(x)$, there is a solution in S at distance at most d. An enumeration problem is d-approximable in polynomial time when S can be generated in total time polynomial in the size of the smallest d-cover of $A(x)$. Allowing randomization to be able to sample the solutions could be important in this context, since we may have much less time than solutions. Remark that a d-approximation algorithm could be extremely useful for a user-interactive system, where the user sets the value d, obtains first a coarse representation of the solutions and can them zoom on an interesting solution by restricting the problem around it and asking for an approximation with a smaller d.

The notion of d-approximability has been designed so that the produced representative solutions should be quite different for the given distance, which hopefully makes such representative solutions useful in practice. A similar approach has been recently proposed through the notion of diversity, which is the sum of the Hamming distances of pairs of elements in a set. A family of hard problems, such as vertex cover, admit FPT algorithms for producing a set of solutions of high enough diversity [11].

Open problem 15. *Give a d-approximation algorithm for a problem with a structured set of solutions, such as the minimal spanning trees or the shortest paths, for any distance over solutions.*

Acknowledgement

The author wants to thank all members of the friendly enumeration community, without your results this column would be empty. Special thanks to Alexandre Vigny and Arnaud Mary for insights I have directly incorporated in the text and to Florent Capelli and Étienne Grandjean for having read this manuscript and suggested many improvements.

References

[1] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

[2] Alfred V Aho and John E Hopcroft. *The design and analysis of computer algorithms*. Pearson Education India, 1974.

[3] Antoine Amarilli, Pierre Bourhis, Louis Jachiet, and Stefan Mengel. A circuit-based approach to efficient enumeration. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 111:1–111:15, 2017.

[4] Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Enumeration on trees with tractable combined complexity and efficient updates. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 89–103. ACM, 2019.

[5] Ricardo Andrade, Martin Wannagat, Cecilia C Klein, Vicente Acuña, Alberto Marchetti-Spaccamela, Paulo V Milreu, Leen Stougie, and Marie-France Sagot. Enumeration of minimal stoichiometric precursor sets in metabolic networks. *Algorithms for Molecular Biology*, 11(1):25, 2016.

[6] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1):21–46, 1996.

[7] Guillaume Bagan. *Algorithms and complexity of enumeration problems for the evaluation of logical queries*. PhD thesis, University of Caen Normandy, France, 2009.

[8] Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *International Workshop on Computer Science Logic*, pages 208–222. Springer, 2007.

[9] Guillaume Bagan, Arnaud Durand, Etienne Grandjean, and Frédéric Olive. Computing the jth solution of a first-order query. *RAIRO-Theoretical Informatics and Applications*, 42(1):147–164, 2008.

[10] Dominique Barth, Olivier David, Franck Quessette, Vincent Reinhard, Yann Strozecki, and Sandrine Vial. Efficient generation of stable planar cages for chemistry. In *International Symposium on Experimental Algorithms*, pages 235–246. Springer, 2015.

[11] Julien Baste, Michael R. Fellows, Lars Jaffke, Tomás Masarík, Mateus de Oliveira Oliveira, Geevarghese Philip, and Frances A. Rosamond. Diversity in combinatorial optimization. *CoRR*, abs/1903.07410, 2019.

[12] Cristina Bazgan, Florian Jamain, and Daniel Vanderpooten. Approximate pareto sets of minimal size for multi-objective optimization problems. *Operations Research Letters*, 43(1):1–6, 2015.

[13] Kateřina Böhmová, Luca Häfliger, Matúš Mihalák, Tobias Pröger, Gustavo Sacomoto, and Marie-France Sagot. Computing and listing st-paths in public transportation networks. *Theory of Computing Systems*, 62(3):600–621, 2018.

[14] Marthe Bonamy, Oscar Defrain, Marc Heinrich, and Jean-Florent Raymond. Enumerating minimal dominating sets in triangle-free graphs. In *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[15] Endre Boros, Khaled Elbassioni, Vladimir Gurvich, and Kazuhisa Makino. Generating vertices of polyhedra and related problems of monotone generation. *Proceedings of the Centre de Recherches Mathématiques at the Université de Montréal, special issue on Polyhedral Computation (CRM Proceedings and Lecture Notes)*, 49:15–43, 2009.

[16] Endre Boros, Vladimir Gurvich, and Peter L Hammer. Dual subimplicants of positive boolean functions. *Optimization Methods and Software*, 10(2):147–156, 1998.

[17] Johann Brault-Baron. *De la pertinence de l'énumération: complexité en logiques propositionnelle et du premier ordre*. PhD thesis, Université de Caen, 2013.

[18] Coen Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[19] Andrei A Bulatov, Víctor Dalmau, Martin Grohe, and Dániel Marx. Enumerating homomorphisms. *Journal of Computer and System Sciences*, 78(2):638–650, 2012.

[20] Florent Capelli and Yann Strozecki. Enumerating models of dnf faster: breaking the dependency on the formula size. *arXiv preprint arXiv:1810.04006*, 2018.

[21] Florent Capelli and Yann Strozecki. Incremental delay enumeration: Space and time. *Discrete Applied Mathematics*, 2018.

[22] Nofar Carmeli and Markus Kröll. On the enumeration complexity of unions of conjunctive queries. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019.*, pages 134–148, 2019.

[23] Katrin Casel, Henning Fernau, Mehdi Khosravian Ghadikolaei, Jérôme Monnot, and Florian Sikora. Extension of some edge graph problems: Standard and parameterized complexity. In *Fundamentals of Computation Theory - 22nd International Symposium, FCT 2019, Copenhagen, Denmark, August 12-14, 2019, Proceedings*, pages 185–200, 2019.

[24] Katrin Casel, Henning Fernau, Mehdi Khosravian Ghadikolaei, Jérôme Monnot, and Florian Sikora. Extension of vertex cover and independent set in some classes of graphs. In *Algorithms and Complexity - 11th International Conference, CIAC 2019, Rome, Italy, May 27-29, 2019, Proceedings*, pages 124–136, 2019.

[25] Frédéric Cazals and Chinmay Karande. A note on the problem of reporting maximal cliques. *Theoretical Computer Science*, 407(1-3):564–568, 2008.

[26] Timothy M Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, 16(4):361–368, 1996.

[27] Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. Generating all maximal induced subgraphs for hereditary and connected-hereditary graph properties. *Journal of Computer and System Sciences*, 74(7):1147–1159, 2008.

[28] Alessio Conte, Roberto Grossi, Andrea Marino, Romeo Rizzi, and Luca Versari. Listing Subgraphs by Cartesian Decomposition. In Igor Potapov, Paul Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, volume 117 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 84:1–84:16, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[29] Alessio Conte, Roberto Grossi, Andrea Marino, and Luca Versari. Sublinear-space bounded-delay enumeration for massive network analytics: Maximal cliques. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 148:1–148:15, 2016.

[30] Alessio Conte, Roberto Grossi, Andrea Marino, and Luca Versari. Listing maximal subgraphs satisfying strongly accessible properties. *SIAM Journal on Discrete Mathematics*, 33(2):587–613, 2019.

[31] Alessio Conte and Takeaki Uno. New polynomial delay bounds for maximal subgraph enumeration by proximity search. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 1179–1190. ACM, 2019.

[32] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[33] Stephen A Cook and Charles W Rackoff. Space lower bounds for maze threadability on restricted machines. *SIAM Journal on Computing*, 9(3):636–652, 1980.

[34] Stephen A. Cook and Robert A. Reckhow. Time bounded random access machines. *J. Comput. Syst. Sci.*, 7(4):354–375, 1973.

[35] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[36] Nadia Creignou and Jean-Jacques Hébrard. On generating all solutions of generalized satisfiability problems. *Informatique théorique et applications*, 31(6):499–511, 1997.

[37] Nadia Creignou, Markus Kröll, Reinhard Pichler, Sebastian Skritek, and Heribert Vollmer. On the complexity of hard enumeration problems. In *International Conference on Language and Automata Theory and Applications*, pages 183–195. Springer, 2017.

[38] Nadia Creignou, Arne Meier, Julian-Steffen Müller, Johannes Schmidt, and Heribert Vollmer. Paradigms for parameterized enumeration. *Theory of Computing Systems*, 60(4):737–758, 2017.

[39] Nadia Creignou and Heribert Vollmer. Parameterized complexity of weighted satisfiability problems: Decision, enumeration, counting. *Fundamenta Informaticae*, 136(4):297–316, 2015.

[40] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[41] Oscar Defrain and Lhouari Nourine. Dualization in lattices given by implicational bases. In *International Conference on Formal Concept Analysis*, pages 89–98. Springer, 2019.

[42] Philippe Duchon, Philippe Flajolet, Guy Louchard, and Gilles Schaeffer. Boltzmann samplers for the random generation of combinatorial structures. *Combinatorics, Probability and Computing*, 13(4-5):577–625, 2004.

[43] Arnaud Durand and Etienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Trans. Comput. Log.*, 8(4):21, 2007.

[44] Arnaud Durand and Yann Strozecki. Enumeration complexity of logical query problems with second-order variables. In *Computer Science Logic (CSL'11)-25th International Workshop/20th Annual Conference of the EACSL*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2011.

[45] Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities. In *European Symposium on Algorithms*, pages 605–617. Springer, 2003.

[46] Hans-Christian Ehrlich and Matthias Rarey. Maximum common subgraph isomorphism algorithms and their applications in molecular science: a review. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 1(1):68–79, 2011.

[47] Thomas Eiter and Georg Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304, 1995.

[48] Thomas Eiter, Georg Gottlob, and Kazuhisa Makino. New results on monotone dualization and generating hypergraph transversals. *SIAM Journal on Computing*, 32(2):514–537, 2003.

[49] David Eppstein et al. K-best enumeration. *Bulletin of EATCS*, 1(115), 2015.

[50] Paul Erdos and Alfred Rényi. On a classical problem of probability theory. *Magyar Tud. Akad. Mat. Kutató Int. Közl*, 6(1-2):215–220, 1961.

[51] Tomás Feder. Network flow and 2-satisfiability. *Algorithmica*, 11(3):291–319, 1994.

[52] Christophe Costa Florêncio, Jonny Daenen, Jan Ramon, Jan Van den Bussche, and Dries Van Dyck. Naive infinite enumeration of context-free languages in incremental polynomial time. *J. UCS*, 21(7):891–911, 2015.

[53] Fedor V Fomin and Dieter Kratsch. *Exact exponential algorithms*. Springer Science & Business Media, 2010.

[54] Michael L Fredman and Leonid Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21(3):618–628, 1996.

[55] Michael L Fredman and Dan E Willard. Surpassing the information theoretic bound with fusion trees. *Journal of computer and system sciences*, 47(3):424–436, 1993.

[56] Itzhak Gilboa and Eitan Zemel. Nash and correlated equilibria: Some complexity considerations. *Games and Economic Behavior*, 1(1):80–93, 1989.

[57] Leslie Ann Goldberg. *Efficient algorithms for listing combinatorial structures*. PhD thesis, University of Edinburgh, UK, 1991.

[58] Petr A Golovach, Pinar Heggernes, Mamadou Moustapha Kanté, Dieter Kratsch, Sigve H Sæther, and Yngve Villanger. Output-polynomial enumeration on graphs of bounded (local) linear mim-width. *Algorithmica*, 80(2):714–741, 2018.

[59] Etienne Grandjean and Thomas Schwentick. Machine-independent characterizations and complete problems for deterministic linear time. *SIAM Journal on Computing*, 32(1):196–230, 2002.

[60] Matthias Hagen. *" Algorithmic and Computational Complexity Issues of MONET*. Cuvillier Verlag, 2008.

[61] Juris Hartmanis and Richard E Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[62] John Hastad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 6–20. Citeseer, 1986.

[63] Mark Jerrum. *Counting, sampling and integrating: algorithms and complexity*. Springer Science & Business Media, 2003.

[64] Chuntao Jiang, Frans Coenen, and Michele Zito. A survey of frequent subgraph mining algorithms. *The Knowledge Engineering Review*, 28(1):75–105, 2013.

[65] David S Johnson, Mihalis Yannakakis, and Christos H Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.

[66] Laurent Juban. Dichotomy theorem for the generalized unique satisfiability problem. In *International Symposium on Fundamentals of Computation Theory*, pages 327–337. Springer, 1999.

[67] Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, and Lhouari Nourine. On the enumeration of minimal dominating sets and related notions. *SIAM Journal on Discrete Mathematics*, 28(4):1916–1929, 2014.

[68] Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, Lhouari Nourine, and Takeaki Uno. A polynomial delay algorithm for enumerating minimal dominating sets in chordal graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 138–153. Springer, 2015.

[69] Dimitris J Kavvadias, Martha Sideri, and Elias C Stavropoulos. Generating all maximal models of a boolean expression. *Information Processing Letters*, 74(3-4):157–162, 2000.

[70] Leonid Khachiyan, Endre Boros, Konrad Borys, Khaled Elbassioni, and Vladimir Gurvich. Generating all vertices of a polyhedron is hard. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 758–765. Society for Industrial and Applied Mathematics, 2006.

[71] Leonid Khachiyan, Endre Boros, Khaled Elbassioni, Vladimir Gurvich, and Kazuhisa Makino. On the complexity of some enumeration problems for matroids. *SIAM Journal on Discrete Mathematics*, 19(4):966–984, 2005.

[72] Donald E Knuth. *The art of computer programming, volume 4A: combinatorial algorithms, part 1*. Pearson Education India, 2011.

[73] Donald Ervin Knuth. *The art of computer programming*, volume 3. Pearson Education, 1997.

[74] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1272–1287. SIAM, 2016.

[75] Eugene L. Lawler, Jan Karel Lenstra, and A. H. G. Rinnooy Kan. Generating all maximal independent sets: Np-hardness and polynomial-time algorithms. *SIAM J. Comput.*, 9(3):558–565, 1980.

[76] Édouard Lucas. *Récréations mathématiques: Les traversees. Les ponts. Les labyrinthes. Les reines. Le solitaire. la numération. Le baguenaudier. Le taquin*, volume 1. Gauthier-Villars et fils, 1882.

[77] Andrea Marino. Enumeration algorithms. In *Analysis and Enumeration*, pages 13–35. Springer, 2015.

[78] Arnaud Mary. *Énumération des Dominants Minimaux d'un graphe*. PhD thesis, Université Blaise Pascal, 2013.

[79] Arnaud Mary and Yann Strozecki. Efficient enumeration of solutions produced by closure operations. In *33rd Symposium on Theoretical Aspects of Computer Science*, 2016.

[80] Arnaud Mary and Yann Strozecki. Efficient enumeration of solutions produced by closure operations. *Discrete Mathematics & Theoretical Computer Science*, 21(3), 2019.

[81] Andrew McGregor. Graph stream algorithms: a survey. *ACM SIGMOD Record*, 43(1):9–20, 2014.

[82] Nimrod Megiddo and Christos H Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, 81(2):317–324, 1991.

[83] John W Moon and Leo Moser. On cliques in graphs. *Israel journal of Mathematics*, 3(1):23–28, 1965.

[84] Shin-ichi Nakano and Takeaki Uno. A simple constant time enumeration algorithm for free trees. *PSJ SIGNotes ALgorithms*, (091-002), 2003.

[85] Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018.

[86] Christos H Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and system Sciences*, 48(3):498–532, 1994.

[87] Christos H Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

[88] Christos H Papadimitriou and Mihalis Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 86–92. IEEE, 2000.

[89] Gara Pruesse and Frank Ruskey. Generating linear extensions fast. *SIAM Journal on Computing*, 23(2):373–386, 1994.

[90] Alexander A Razborov. Lower bounds for the monotone complexity of some boolean functions. In *Soviet Math. Dokl.*, volume 31, pages 354–357, 1985.

[91] RC Read and RE Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5(3):237–252, 1975.

[92] Frank Ruskey. Combinatorial generation. *Preliminary working draft. University of Victoria, Victoria, BC, Canada*, 11:20, 2003.

[93] Luc Segoufin. Constant delay enumeration for conjunctive queries. *SIGMOD Record*, 44(1):10–17, 2015.

[94] Yann Strozecki. *Enumeration complexity and matroid decomposition*. PhD thesis, Université Paris Diderot - Paris 7, 2010.

[95] Yann Strozecki. On enumerating monomials and other combinatorial structures by polynomial interpolation. *Theory of Computing Systems*, 53(4):532–568, 2013.

[96] James C Tiernan. An efficient search algorithm to find the elementary circuits of a graph. *Communications of the ACM*, 13(12):722–726, 1970.

[97] Takeaki Uno. Constant time enumeration by amortization. In *Workshop on Algorithms and Data Structures*, pages 593–605. Springer, 2015.

[98] Sergei Vassilvitskii and Mihalis Yannakakis. Efficiently computing succinct trade-off curves. *Theoretical Computer Science*, 348(2-3):334–356, 2005.

[99] Alexandre Vigny. *Query enumeration and nowhere dense graphs*. PhD thesis, Université Paris-Diderot, 2018.

[100] Timothy Walsh. Generating gray codes in o (1) worst-case time per word. In *International Conference on Discrete Mathematics and Theoretical Computer Science*, pages 73–88. Springer, 2003.

[101] Kunihiro Wasa. Enumeration of enumeration algorithms. *arXiv preprint arXiv:1605.05102*, 2016.

[102] Avi Wigderson. The complexity of graph connectivity. In *International Symposium on Mathematical Foundations of Computer Science*, pages 112–132. Springer, 1992.

[103] Ryan Williams. Nonuniform acc circuit lower bounds. *Journal of the ACM (JACM)*, 61(1):2, 2014.

[104] Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, 2018.

[105] Robert Alan Wright, Bruce Richmond, Andrew Odlyzko, and Brendan D McKay. Constant time generation of free trees. *SIAM Journal on Computing*, 15(2):540–548, 1986.

[106] Takayuki Yato and Takahiro Seta. Complexity and completeness of finding another solution and its application to puzzles. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 86(5):1052–1060, 2003.

[107] R. Zippel. Interpolating polynomials from their values. *Journal of Symbolic Computation*, 9(3):375–403, 1990.