UNIVERSITE DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Laboratoire Données et Algorithmes pour une Ville
Intelligente et Durable David – UR 7431

HABILITATION A DIRIGER DES RECHERCHES

DISCIPLINE / SPECIALITE :   INFORMATIQUE / ALGORITHMIQUE

Présentée par :

Yann STROZECKI

**Enumeration Complexity: Incremental Time, Delay and
Space**

Soutenue le : DATE

JURY

Mme Nadia Creignou, Professeur, Université d'Aix Marseille (Rapporteur)

Mr Reinhard Pichler, Professeur, Technische Universität Wien (Rapporteur)

Mr Takeaki Uno, Professeur, National Institute of Informatics (Rapporteur)

Mr Louhari Nourine, Université de Clermont Auvergne (Examinateur)

Mme Marie-France Sagot, Directirce de Recherche, INRIA Rhône-Alpes (Examinateur)

Mr Jean-Michel Fourneau, Professeur, Université de Versailles Saint-Quentin-en-Yvelines (Tuteur)

Acknowledgments

Merci à mes enfants de m'avoir permis de ne pas trop me précipiter pour écrire mon habilitation. Merci au confinement pour m'avoir permis de prendre encore plus de recul et moins de RER. Merci à ma compagne de m'avoir épousé pendant que je rédigeais ce manuscrit, j'espère pouvoir encore continuer longtemps à équilibrer ma vie familiale et professionelle ainsi.

Je suis très reconnaissant à tous les collègues qui m'ont proposé des problèmes, invité à des séminaires et des conférences et m'ont posé de nombreuses questions: c'est grâce à vous j'ai continué à travailler sur l'énumération. Je suis également reconnaissant à tous mes coauteurs, avec qui j'ai eu beaucoup de plaisir à travailler sur les résultats présentés dans ce manuscrit. Je suis prêt pour des nouvelles questions Arnaud et Florent, il va falloir trouver une garde pour vos enfants ! Thanks to Olivier Commowick for distributing a nice thesis template (at http://olivier.commowick.org/thesis_template.php).

I heartily thank the reviewers, Nadia Creignou, Reinhard Pichler and Takeaki Uno to have accepted to take my manuscript as a holiday reading.

Contents

Introduction

1.1 Foreword

During the past ten years, I have worked in several areas related to algorithms and complexity. I was first interested in algebraic complexity and computer algebra during a post-doc in Toronto, working with Bruno Grenet, Pascal Koiran and Natacha Portier. Then, I began to study algorithmic game theory with David Auger, Pierre Coucheney and our PhD student Xavier Badin de Montjoye, working on algorithms to find optimal strategies for simple stochastic games. More recently, I have studied periodic scheduling problems with my PhD student Maël Guiraud, both from a theoretical and practical point of view, to optimize the latency of telecommunication networks.

In this habilitation thesis, I focus on a fourth topic I have studied since the beginning of my PhD: the structural complexity of enumeration, in particular the relationships between different notions of tractability. This thesis tries to be a survey on the complexity of enumeration, with an emphasis on my contributions to the area. A good part of this thesis is taken from my recent survey [Strozecki 2019] and some of my articles on the complexity of enumeration [Capelli & Strozecki 2019, Mary & Strozecki 2019, Capelli & Strozecki 2021a]. I found inspiration and a broad view on enumeration in several theses [Bagan 2009, Strozecki 2010b, Brault-Baron 2013, Mary 2013, Marino 2015, Vigny 2018]. I was also motivated to write a survey on enumeration complexity and this thesis by the ever increasing community of researchers interested in enumeration coming from many different backgrounds: graph algorithms, parametrized complexity, exact exponential algorithms, logic, databases, enumerative combinatorics, applied algorithms in bioinformatics, cheminformatics, networks ... In the last ten years, the topic has attracted much attention and these different communities began to share their ideas and problems, as exemplified by the creation of WEPA, the International Workshop on Enumeration Problems and Applications, two recent Dagstuhl worskhops *"Algorithmic Enumeration: Output-sensitive, Input-Sensitive, Parameterized, Approximative"* and *"Enumeration in Data Management"*, and the creation of wikipedia pages for enumeration complexity and algorithms. The results I present have been obtained with my co-authors, in particular Florent Capelli, Arnaud Mary, Arnaud Durand, Sandrine Vial and Franck Quessette but they also originate from helpful discussions with my colleagues Nadia Creignou, Frédéric Olive, Étienne Grandjean, Stefan Mengel, Mamadou Kanté, Lhouari Nourine, Alexandre Vigny, Florent Madelaine and many others.

1.2   Introduction

Modern enumeration algorithms date back to the 70's with graph algorithms [Tiernan 1970], while fundamental complexity notions for enumeration have been proposed 30 years ago by Johnson, Yannakakis and Papadimitriou [Johnson *et al.* 1988]. However, much older problems can be reinterpreted as enumeration: the baguenaudier game [Lucas 1882] 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 [Ruskey 2003] 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 [Knuth 2011] and Knuth even confessed that these are his favorite algorithms; while I agree I would extend that appreciation to all enumeration algorithms.

Hundreds of different enumeration problems have now been studied, see the actively maintained list Enumeration of Enumeration Algorithms and Its Complexity [Wasa 2016]. Some enumeration problems have important practical applications: A database query is the enumeration of assignments of a formula, data mining relies on generating all frequent objects in a large database (frequent itemsets [Agrawal *et al.* 1994] and frequent subgraphs [Jiang *et al.* 2013]), finding the minimal transversals of a hypergraph has applications in many fields [Hagen 2008] such as biology, machine learning, cryptography . . .

A classical approach to enumeration is to see it as a variation on decision or search problems 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 (maximal) matching.

- (optimization problem) Produce a matching of largest cardinality.

- (counting problem) Count all matchings.

- (enumeration problem) List all (maximal) 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 problems is that the output is usually very large with regard to the input. Hence, interpreting the *total time* to produce the whole set of solutions as a function of the input size is not informative, since it is almost always exponential. To make enumeration complexity relevant, we need to consider other parameters of the problem out of the instance size: 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. We must also study finer complexity measures than total time, since it does not allow to differentiate most enumeration problems.

The simplest way to improve on the analysis of enumeration algorithms is to evaluate how the *total time* to compute all solutions relates 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* [Fomin & Kratsch 2010]. 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 bounds, for instance on the convex hull problem [Chan 1996]. 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 in 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, such as the generation of unrooted trees of a given size [Wright *et al.* 1986], linear extensions of a partial order [Pruesse & Ruskey 1994] or integers given in Gray code order [Knuth 1997]. Uno also proposed in [Uno 2015] 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 [Ehrlich & Rarey 2011]. 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 [Andrade *et al.* 2016, Barth *et al.* 2015, Böhmová *et al.* 2018]. In particular, when confronted to a multicriteria optimisation problem, a natural approach is to enumerate the Pareto's frontier [Papadimitriou & Yannakakis 2000, Vassilvitskii & Yannakakis 2005, Bazgan *et al.* 2015]. 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 [Khachiyan *et al.* 2005] and is thus in polynomial incremental time. The fundamental problem of generating the minimal transversals of a hypergraph can also be solved in quasi-polynomial incremental time [Fredman & Khachiyan 1996] and some of its restrictions in polynomial incremental time [Eiter *et al.* 2003].

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 [Read & Tarjan 1975], the satisfying assignments of some tractable variants of SAT [Creignou & Hébrard 1997] or the spanning trees and connected induced subgraphs of a graph [Avis & Fukuda 1996]. All polynomial delay algorithms are based on few methods such as *backtrack search* (also called flashlight search or binary partition) or *reverse search*, see [Mary 2013] for a survey.

When the size of the input is much larger than the size of one solution, think of generating subsets of vertices of a hypergraph or a small query 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 [Golovach *et al.* 2018].

Unlike classical complexity classes, none of the classes introduced in this thesis but EnumP, the equivalent of NP, have complete problems. Because of that, no notion of reduction for enumeration seems better than the others, and for many proof of hardness, ad hoc reductions are used. To overcome the lack of completness result, several works restrict themselves to smaller families of enumeration problems in the hope of better classifying their complexity: assignments of SAT formula [Creignou & Hébrard 1997], homomorphisms [Bulatov *et al.* 2012], subsets given by saturation operators [Mary & Strozecki 2019], FO queries over various structures [Segoufin 2015], maximal subgraphs [Cohen *et al.* 2008, Conte & Uno 2019, Conte *et al.* 2019]...

Organization In Chapter 2, enumeration problems and the related computation model are defined, with an emphasis on the consequences of several definitional choices. Then, in Chapter 3, we introduce complexity classes related to three time complexity measures, *total time*, *incremental time* and *delay*. For each of these classes, we provide a separation theorem and a characterization when possible. In Chapter 4, we study several restrictions on space in enumeration. In particular, we show that incremental linear time is in fact equal to polynomial delay, even with polynomial space. Chapter 5 is devoted to the presentation of low complexity classes, with strong contraints on the delay, to capture tractability in different contexts. In particular, we present several methods to show that a problem has a strong polynomial delay, applied to the problem of generating the models of a DNF formula. We also investigate the restricted class of problems which admits a uniform random generator of solutions, as a way to avoid using space in addition of time, and show that it can be related to polynomial delay. Instead of generating solutions in a random order or an unspecified one, it is often relevant to generate them in a specified order, to get more "interesting" solutions first. We review how an additional constraint on the order of generated solutions may increase the complexity in Chapter 6. In Chapter 7, we review the few lower bound results for enumeration problems. To obtain more lower bounds and classification results, we present several restricted frameworks from logic and algebra. In addition to lower bounds, we provide many algorithms for tractable classes of saturation problems, interpolation problems and generalized first order queries. In Chapter 8, we briefly present an algorithm developed to enumerate a certain kind of planar map useful in cheminformatic, underlying the differences between the design of a theoretical

algorithm and the implementation of an algorithm used in practice. To deal with a huge solution space, even constant delay algorithm are not satisfactory, hence, in Chapter 9, we present several alternative approaches to the task of listing solutions exhaustively.

Enumeration Framework

Contents

2.1 Enumeration Problem

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 ENUM·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 cardinality of a set S by $|S|$.

Example 1. Let $H = (X, E)$ be an hypergraph, with X the set of vertices and E the set of hyperedges. A transversal or hitting set is a subset T of X, such that all hyperedges $e \in E$ have a non-empty intersection with T. In other words, T is a cover of the hypergraph H. The problem of generating all transversals minimal for inclusion is a fundamental enumeration problem, with many applications [Hagen 2008].

The binary predicate MIN-TRANSVERSALS(H, T) is true if and only if T is a minimal transversal of H. The problem of listing minimal transversals is denoted by ENUM·MIN-TRANSVERSALS. Hypergraph H is an input of the problem, MIN-TRANSVERSALS(H) denotes the set of all minimal transversals of H and $T \in$ MIN-TRANSVERSALS(H) is called a solution of ENUM·MIN-TRANSVERSALS (for input H).

In this thesis, 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 [Florêncio *et al.* 2015].

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 size. 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.

Definition 1. *A binary predicate A is* polynomially balanced *if there is a polynomial p, such that, for all $y \in A(x)$, $|y| \leq p(|x|)$.*

Let CHECK·A be the problem of deciding, given x and y, whether $y \in A(x)$. In almost all practical enumeration problems, one can check efficiently whether a string is a solution. This can be captured by the constraint CHECK·$A \in \mathsf{P}$ From a polynomially balanced predicate A with CHECK·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 of enumeration problems from these predicates, analogous to NP.

Definition 2. *The class* EnumP *is the set of all problems* ENUM·A *where A is polynomially balanced and* CHECK·$A \in \mathsf{P}$.

Example 2. Any transversal T of an hypergraph $H = (X, E)$ satisfies $T \subseteq X$, hence $|T| \leq |H|$ and MIN-TRANSVERSALS is polynomially balanced. Moreover, CHECK·MIN-TRANSVERSALS can be solved in polynomial time in H. First we test whether T covers all edges of E and then we verify it is minimal for this property, that is $T \setminus \{v\}$ is not a transversal for all v vertices of H. Hence, ENUM·MIN-TRANSVERSALS $\in$ EnumP.

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 have some complete problems for the simple parsimonious reduction. However, there is no standard notion of reduction which makes EnumP-complete all seemingly hard problems of EnumP, and this difficulty is already apparent in the definition of the polynomial hierarchy for enumeration. This hierarchy is built in [Creignou *et al.* 2017a] by adding oracles in the polynomial hierarchy to polynomial delay or incremental polynomial time machine. This gives a strict hierarchy of hard problems, with some natural complete examples in its first levels. Note that, in contrast to counting complexity, defining a hierarchy through the complexity of CHECK·A does not seem relevant, since it is easy to define a "simple" enumeration problem with a hard CHECK·A problem by adding many trivial solutions to the problem.

Finally, note that in the definition of EnumP nothing specific about enumeration is taken into account. We are able to define it before even specifying the computation model or complexity measures specific to enumeration, as it only relies on the complexity of deciding the auxiliary problem CHECK·A.

2.2   Model of Computation

The model of computation is the random access machine (RAM) with comparison, addition, subtraction and multiplication as its basic arithmetic operations and an operation $\texttt{Output}(i, j)$ which outputs the concatenation of the values of registers $R_i, R_{i+1}, \ldots, R_j$. RAM machines have been introduced by Cook and Reckhow to better model the storage of existing machines [Cook & Reckhow 1973, Aho & Hopcroft 1974]; for variants designed for enumeration see [Bagan 2009, Strozecki 2010b]. 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 $(y_1, \ldots, y_s)$ such that $A(x) = \{y_1, \ldots, y_s\}$ 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. 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 up to point when 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 $(y_1, \ldots, y_s)$ is the maximum over all $i \leq s$ of the time the machine uses between the generation of y_i and y_{i+1}, that is $\max_{1 \leq i < s} T(x, i+1) - T(x, i)$. 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 in this thesis that *there is no postprocessing*: enumeration stops as soon as the last solution is output. To make low complexity classes interesting, it is important to allow preprocessing, that is $T_M(x, 1)$, to be larger than the delay and we will mention it when appropriate. The *average delay* or *amortized time* of a RAM machine is the average time needed by the machine to produce a solution, that is $T(x, s)/s$. The delay is an upper bound to the average delay.

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 [Cook & Reckhow 1973, Papadimitriou 2003]. 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 in enumeration such as AVL trees or tries [Cormen *et al.* 2009] which gives linear time access to elements 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 the size of the solution output, in particular a constant delay. In all definitions of RAM machine for enumeration [Bagan 2009, Strozecki 2010b, Mary 2013, Brault-Baron 2013] 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 [Papadimitriou 2003]. Constant time output is meaningful, when only the deltas between solutions are output rather than solutions themselves. It is also relevant, if we just do a constant time operation on each solution such as counting them or evaluating some measure which depends only on the constant amount

---

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

of changes between two consecutive solutions.

The originality of the RAM model proposed in this thesis is to allow outputing 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 a computer, the memory zone in which solutions are stored is not always the same. In fact, a programmer does not even control (but rather the system and the cpu) where information is physically stored in memory. Our model enables us 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 allow unbounded integers in registers 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 [Cook & Reckhow 1973, Aho & Hopcroft 1974]) 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 [Grandjean & Schwentick 2002] and constant delay in enumeration [Durand & Grandjean 2007, Bagan 2009]. It is similar to the word RAM model, a transdichotomous model [Fredman & Willard 1993], used to give finer and more realistic bounds for data structures. In some enumeration algorithms, all solutions must be stored and there 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.

2.3   Parsimonious Reduction

In this section we introduce a first notion of reduction for enumeration problems. A reduction is a binary relation over enumeration problems (or equivalently over the relations which define the problems). A reduction, denoted by $\prec$, must be *transitive*, that is if $\textsc{Enum}{\cdot}A \prec \textsc{Enum}{\cdot}B$ and $\textsc{Enum}{\cdot}B \prec \textsc{Enum}{\cdot}C$ then $\textsc{Enum}{\cdot}A \prec \textsc{Enum}{\cdot}C$. A class of enumeration problems $\mathcal{C}$ is closed under the reduction $\prec$ if $\textsc{Enum}{\cdot}B \in \mathcal{C}$ and $\textsc{Enum}{\cdot}A \prec \textsc{Enum}{\cdot}B$ imply $\textsc{Enum}{\cdot}A \in \mathcal{C}$. This property guarantees that reductions can be used to provide algorithms and not just hardness results. Note that if a reduction allows for too much computation with regard to a

class, the class is not closed under the reduction, for instance P is not closed under exponential time reductions.

The *parsimonious reduction* for counting problems enforces equality of numbers of solutions, by requiring a bijection between sets of solutions in addition to the bijection between inputs. If this bijection is explicit and tractable, then such a reduction is adapted to enumeration complexity.

Definition 3 (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, $g(x, \cdot)$ *restricted to* $B(f(x))$ *is a bijection with* $A(x)$ *and* $g(x, \cdot)^{-1}$ *is polynomial time computable.*

Example 3. Let CLIQUE(G, S) be the predicate which is true when S is a clique in the graph G. Let INDEPENDENT-SET(G, S) be the predicate which is true when S is an independent set in the graph G. Let f be the function which maps the graph G to its complement graph H, that is two vertices are adjacent in G if and only if they are not adjacent in H. Let $g(G, S)$ be the function which maps S a set of vertices in G to the same set of vertices S in $f(G)$. Using these two functions computable in polynomial time, we have ENUM·CLIQUE $\prec$ ENUM·INDEPENDENT-SET (and also ENUM·INDEPENDENT-SET $\prec$ ENUM·CLIQUE).

Let us denote by $g_x(y)$ the bijection beetween $B(f(x))$ and $A(x)$ in the definition of parsimonious reduction. The condition that g_x^{-1} is polynomial time computable is in the definition only to ensure that EnumP is stable under parsimonious reduction as we now prove. Recall that ENUM·$B \in$ EnumP means that B is polynomially balanced and CHECK·B is in polynomial time. To prove that EnumP is closed under parsimonious reductions, we must prove that if ENUM·$A \prec$ ENUM·B and ENUM·$B \in$ EnumP, then ENUM·$A \in$ EnumP. The predicate $A(x, y)$ holds if and only if $B(f(x), g_x^{-1}(y))$, hence it can be decided in polynomial time because we have assumed that B, f and g_x^{-1} are polynomial time computable. Second, since f and g are polynomial time computable, the elements of $A(x) = g(x, B(f(x)))$ are polynomial in the size of x.

An EnumP-complete problem is defined as a problem in EnumP to which any problem in EnumP reduces by parsimonious reduction. 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 3SAT is NP-complete [Cook 1971] is parsimonious.

Let us consider the predicate $3SAT_0(\phi, x)$ which is true if and only if x is a satisfying assignment of the 3-CNF formula ϕ or x is the all zero assignment. Then $3SAT_0(\phi)$ is never empty and therefore many problems of EnumP cannot be reduced to ENUM·$3SAT_0$ by parsimonious reduction. However, the problem intuitively feels like a complete problem and it can be made so by relaxing the reduction. Indeed, from an enumeration perspective, ENUM·$3SAT_0$ is exactly as hard as ENUM·$3SAT$: given an algorithm to enumerate ENUM·$3SAT_0$, we can enumerate ENUM·$3SAT$ by discarding if necessary the all zero solution (or adding it when doing the inverse reduction). This reduction requires an additional polynomial time computation only. This shows that parsimonious reductions are not sufficient to classify the complexity of enumeration problems, and we present several other reductions in the next chapter.

Time Complexity: Total Time, Incremental Time and Delay

Contents

3.1   Output Polynomial Time

To measure the complexity of an enumeration problem, we consider the total time taken to compute all solutions. If the total time of an algorithm depends on the size of the input only, we speak of *input sensitive* algorithm. The Bron-Kerbosch algorithm [Bron & Kerbosch 1973] finds all maximal cliques in a graph in time $O(2^{n/3})$, with n the number of vertices of the graph. Since there can be as many as $2^{n/3}$ maximal cliques in a graph of $n/3$ disjoint triangles, Bron-Kerbosch algorithm is optimal with regard to the input size. However, its complexity does not decrease with the number of solutions, or at least there is no proof of such result. Since the number of solutions is much smaller than $2^{n/3}$ for most instances, it is more precise and relevant to bound the total time as a function of the size of the input and of *the output*. By contrast to input sensitive algorithms, such algorithms are called *output sensitive*. Note that bounding the total time by a polynomial in the input size only is not relevant, since it forces the number of solutions to be polynomial. Then, to define efficient algorithms, it is natural to bound the total time by a polynomial in the number of solutions and in the size of the input. Algorithms with this complexity are said to be in **output polynomial time** or sometimes in polynomial total time.

Definition 4 (Output polynomial time). *A problem* ENUM·$A \in$ EnumP *is in* OutputP *if there is a polynomial $p(x, y)$ and a machine M which solves* ENUM·A *and such that for all x, $T(x, |A(x)|) < p(|x|, |A(x)|)$.*

For instance, if we represent a polynomial by its set of monomials, then classical algorithms for interpolating multivariate polynomials from their values are output polynomial [Zippel 1990, Strozecki 2010a] as they produce the polynomial in time proportional to the number of its monomials, see Chapter 7 for more details.

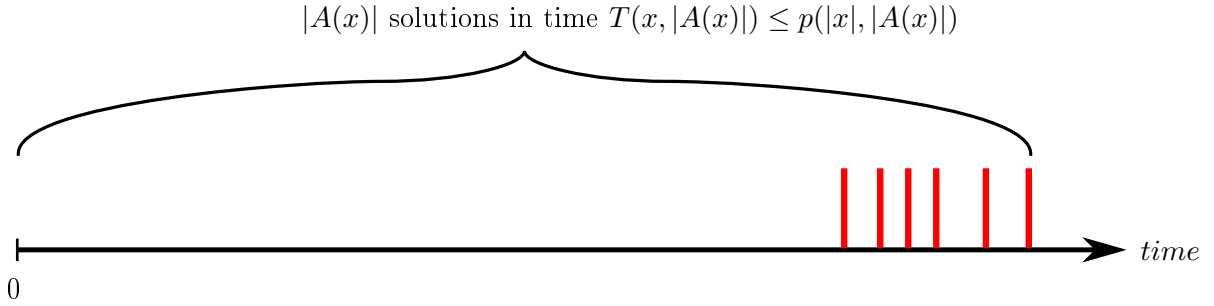$|A(x)|$ solutions in time $T(x, |A(x)|) \leq p(|x|, |A(x)|)$

Figure 3.1: An enumeration in output polynomial time. A red bar represents the output of a solution at some point in time of the enumeration.

The classes EnumP and OutputP may be seen as analog of NP and P for enumeration. It turns out that their separation is equivalent to the P = 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 [Capelli & Strozecki 2019]). OutputP = EnumP *if and only if* P = NP.

Output polynomial algorithms are useful when all solutions need to be generated, for a proof of optimality or to build an exhaustive library of objects. However, if we want to generate only a few solutions, because the whole process is too long, an output polynomial algorithm does not offer any guarantee, since all solutions can be produced at the end of the computation, see Figure 3.1.

3.2   Incremental Polynomial Time

When generating all solutions is too long, we want to be able to generate at least some. To do so, we can fix an arbitrary parameter k and ask to enumerate only k solutions. Sometimes, an order is fixed on the solutions and the problem is to find the *k-best* ones. The k-shortest path and the k-minimum spanning tree problems are typical k-best problems with many applications as explained in the survey [Eppstein *et al.* 2015]. In the enumeration paradigm, we are slightly more general: we do not require a parameter k (either fixed or given as input), but want to measure and bound the time used by an algorithm to produce any 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 $f(k)g(n)$ and preprocessing $h(n)$ if on every input x of size n, we have:

- $T(x, 1) \leq h(n)$ (preprocessing)

- for all $k \leq |A(x)|$, $T(x, k) - T(x, 1) \leq f(k)g(n)$ (incremental time)

Separating the preprocessing from the rest of the computation allows to set up expensive data structures or to simplify the input at the beginning of an enumeration. In that way,

classes with small incremental time contain more interesting problems and we better represent the time to enumerate many solutions, which otherwise may be artificially hidden by the preprocessing time. When the preprocessing is dominated by the incremental time, that is $h(n) \leq f(1)g(n)$, we omit it.

Definition 5 (Incremental polynomial time). *Let $a \geq 1$. A problem $\text{ENUM} \cdot A \in \text{EnumP}$ is in IncP_a if there is a constant b, a polynomial p and a machine M which solves $\text{ENUM} \cdot A$ on instances of size n in incremental time $O(k^a n^b)$ and preprocessing time $O(p(n))$. Moreover, we define $\text{IncP} = \bigcup_{a \geq 1} \text{IncP}_a$.*

Note that allowing or not arbitrary polynomial time preprocessing does not modify the class IncP, since this preprocessing can always be interpreted as a polynomial time before outputing the first solution. We require that $\text{ENUM} \cdot A \in \text{EnumP}$ so that $\text{ENUM} \cdot A \in \text{IncP}$, however there are a few examples of problems, whose solutions cannot be tested in polynomial time, but with an algorithm in incremental polynomial time. A clause sequence is the set of clauses satisfied by some assignment of a CNF formula, hence deciding whether a formula admits the clause sequence with all clauses is NP-complete. However, there is a clever incremental time algorithm to generate all clause sequences for k-CNFs [Bérczi *et al.* 2021]. It relies on finding enough simple to enumerate solutions, to have the time to divide the problem into many simpler subformulas with few sequences in common.

Let A be a binary predicate, AnotherSol_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 $A(x) \subseteq \mathcal{S}$ (see [Strozecki 2010b, Creignou *et al.* 2017a]). The problems in IncP are the ones with a polynomial time search problem, as stated in the following proposition.

Proposition 3.2 (Folklore, see [Capelli & Strozecki 2019]). *Let A be a predicate such that $\text{ENUM} \cdot A \in \text{EnumP}$. AnotherSol_A is in FP if and only if $\text{ENUM} \cdot A$ is in IncP.*

Recall that the **delay** of producing the ith solution by some algorithm, is the time between the production of two consecutive solutions, that is, for $1 \leq k < |A(x)|$, $|T(x, k+1) - T(x, k)|$. 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 6 (Usual definition of incremental polynomial time). *Let $a \geq 0$. A problem $\text{ENUM} \cdot A \in \text{EnumP}$ is in UsualIncP_a if there is a constant b, a polynomial p and a machine M which solves $\text{ENUM} \cdot A$ on instances of size n, with preprocessing time $O(p(n))$ and such that $|T(x, k+1) - T(x, k)| \in O(k^a n^b)$.*

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 computing the accessible vertices from a source in a graph by flooding, computing a finite group from its generators, enumerating the circuits of matroids [Khachiyan *et al.* 2005] or computing a closure by set operations [Mary & Strozecki 2016a].

Example 4. Let UNION be the binary predicate such that $\text{UNION}(x, y)$ is true if and only if x is a set $\{s_1, \ldots, s_m\}$ of subsets of $[n]$ and if $y \subseteq [n]$ can be obtained by union of elements in x, i.e. there is $I \subseteq [m]$, such that $y = \cup_{i \in I} s_i$.

ENUM·UNION can be easily solved by a saturation algorithm, where the production rule is the union of two solutions and the set of solutions is initialized by x. As an example, consider $x = \{\{1,2\},\{2,3\},\{1,4\},\{1,3,4\}\}$. The production rule successively adds $\{1,2\} \cup \{2,3\}$, $\{1,2\} \cup \{1,4\}$, $\{1,2\} \cup \{1,3,4\}$ then the algorithm stops since no union of the obtained solutions gives a new solution. This algorithm proves that ENUM·UNION $\in$ IncP$_2$, since for each new solution output we need to check its union with all other solutions.

With our definition of incremental polynomial time, 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 queue and outputing them regularly from the queue.

Proposition 3.3 (Folklore, see [Capelli & Strozecki 2019]). *For all* $a \geq 0$, IncP$_{a+1}$ = UsualIncP$_a$.

The amortization method plus a data structure to detect duplicates such as a trie, allow to deal with bounded repetitions of solutions and is often used to simplify the design of enumeration algorithms, see for instance the well-named Cheater's Lemma [Carmeli & Kröll 2019].

Since IncP is characterized by the search problem AnotherSol$_A$, it can be related to the class TFNP introduced in [Megiddo & Papadimitriou 1991]. 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 NP $\cap$ 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 AnotherSol$_A$ into a problem of TFNP, so that, assuming TFNP = FP, a solution to this problem is found which can then be used to solve AnotherSol$_A$. Conversely, a TFNP problem can be turned into an enumeration problem in OutputP, by padding.

Proposition 3.4 ([Capelli & Strozecki 2019]). TFNP = FP *if and only if* IncP = 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 3.4. However, no natural problem is known to be inside OutputP but not in IncP.

Open problem 3.1. *Find a natural problem in* OutputP *but not in* IncP. *It should be a problem for which* AnotherSol *is* TFNP-*hard because of Proposition 3.4, but the decision version of* AnotherSol, *which asks whether another solution exists, should be in* P. *One candidate problem is the enumeration of minimal dominating sets in* K_t-*free graphs, which is in* OutputP *[Bonamy et al. 2020] but is not known to be in* IncP.

We can get a finer separation of classes than in Proposition 3.4: (IncP$_a$)$_{a\in\mathbb{R}^+}$ form a strict hierarchy inside IncP. We need to assume some complexity hypothesis since P = NP implies IncP = IncP$_1$. Because we need to distinguish between different polynomial complexities as in fine grained complexity [Williams 2018], we also rely on the Exponential Time Hypothesis (ETH). It states that there exists $\varepsilon > 0$ such that there is no algorithm for 3-SAT in time $\tilde{O}(2^{\varepsilon n})$ where n is the number of variables of the formula and $\tilde{O}$ means that factors in $n^{O(1)}$ are hidden.
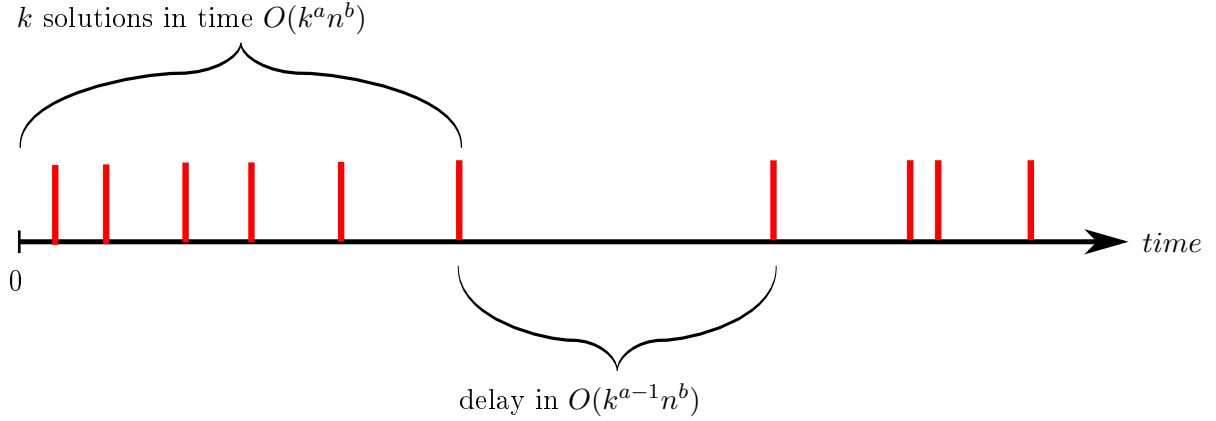
Figure 3.2: An enumeration in incremental polynomial time. Delay guaranteed when Proposition 3.3 is used.

Theorem 3.1 ([Capelli & Strozecki 2019]). *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 repeatedly proves the theorem. With a similar but simpler proof, the same strict hierarchy for OutputP can be obtained.

Open problem 3.2. *Prove the converse of Theorem 3.1 or weaken the* ETH *hypothesis.*

In several applications, in particular when enumerating the result of a query over a database, it is asked to output the solutions regularly, so to process those solutions on the fly without storing them nor pausing the enumeration process. Incremental polynomial time guarantes only a limited form of regularity as illustrated in Figure 3.2, which does not seem sufficient.

3.3 Polynomial Delay

If we want to have a regular enumeration process, then the delay between solutions should not change along the enumeration. The **delay of a machine**, on input x, is the maximum of the delay between two solutions it produces, that is $\max_{1 \leq k < |A(x)|} |T(x, k+1) - T(x, k)|$. In this section, we bound the delay by a polynomial in the input size, but we could require a stronger bound, a theme which is explored in Chapter 5.

Definition 7 (Polynomial delay). *A problem* ENUM·$A \in$ EnumP *is in* DelayP *if it is solved by a machine* M *with preprocessing and delay polynomial in its input size.*

Observe that, by definition, DelayP = UsualIncP$_0$ and is thus equal to IncP$_1$ and included in IncP by Proposition 3.3. Polynomial delay is the most usual notion of tractability in enumeration, both because it is a strong property (regularity and linear total time) and because
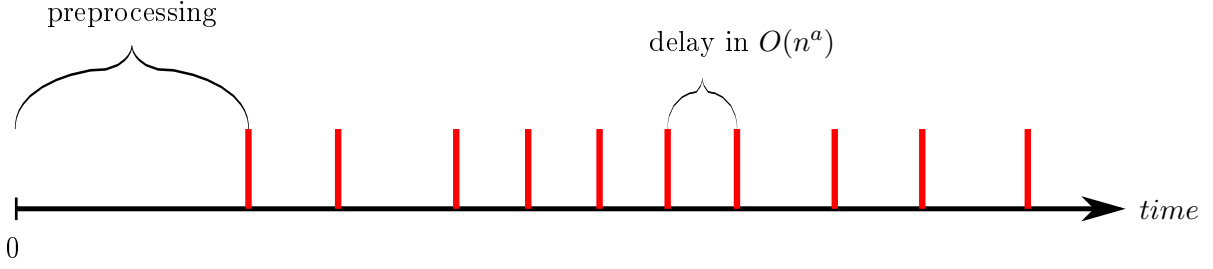
Figure 3.3: An enumeration in polynomial delay.

it is relatively easy to obtain. Indeed, most methods used to design enumeration algorithms such as backtrack search with an easy extension problem [Mary & Strozecki 2019], or efficient traversal of a supergraph of solutions [Lawler *et al.* 1980, Avis & Fukuda 1996], yield polynomial delay algorithms when they are applicable. Algorithms in DelayP have the same total time and the same incremental time as algorithms in IncP_1, but they enumerate solutions more regularly as shown in Figure 3.3. We further study in Chapter 4 the relationship between DelayP and IncP_1, when the space is polynomially bounded.

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 y, whether there is y' such that $yy' \in A(x)$. Assuming the alphabet is $\{0,1\}$, then the set $A(x)$ can be divided into solutions beginning by 0 and solutions beginning by 1. Then, ExtSol·A can be used to decide whether any of these subsets of solutions is empty. If ExtSol·$A \in \mathsf{P}$, then this method of partitionning the set of solutions, applied recursively on non-empty sets of solutions is an efficient version of backtrack search, often called *binary partition* (see e.g. [Mary & Strozecki 2019]). In this manuscript we call this method the *flashlight search*, the flashlight stands for the use of ExtSol·A which "illuminates" a whole subtree of the search tree to discard it if it contains no solution.

Proposition 3.5. *If* ExtSol·$A \in \mathsf{P}$, *then* Enum·$A \in$ DelayP.

Example 5. As an illustration of the flashlight search, we use it to solve Enum·Union. Let us assume that a set over n element is given by its characteristic vector. Let y be a partial characteristic vector, which can be represented by A the set of positions of 1 in y (elements in the set represented by y) and B the set of positions of 0. Extending y such that $yy' \in$ Union(x) can be recast as finding a set $S \in$ Union(x) such that $A \subseteq S$ and $S \cap B = \emptyset$. Let $x^B = \cup_{s \in x, \, s \cap B = \emptyset} s$, by construction it is the largest set of Union(x) which has an empty intersection with B. Hence, ExtSol·Union(x, y) is true if and only if $A \subseteq x^B$, and ExtSol·Union can thus be solved in time $O(mn)$, when x is a set of m subsets of $[n]$. The enumeration algorithm is a recursive algorithm, which does a depth first traversal of the implicit tree of partial solutions, as represented in Figure 3.4. The delay of the algorithm is the depth of the tree times the complexity of solving ExtSol·Union, that is $O(mn^2)$ but it can be improved to $O(mn)$ by amortizing the cost of solving ExtSol·Union on a branch, see [Mary & Strozecki 2019].
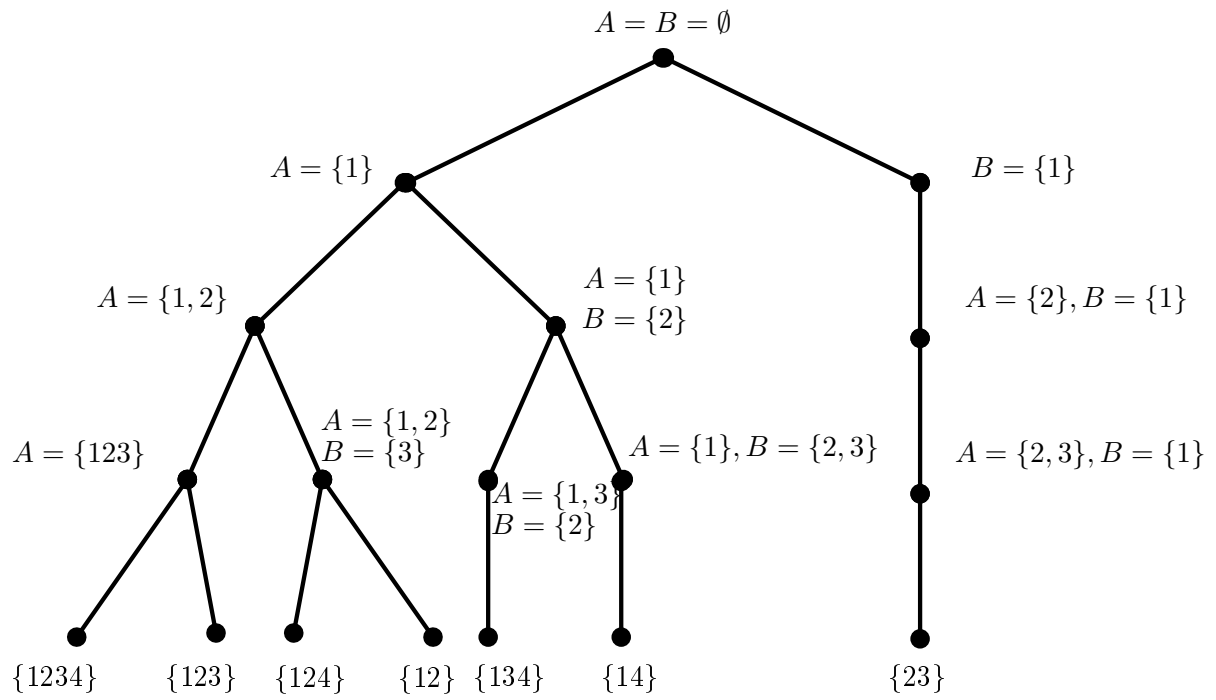
Figure 3.4: The tree of partial solutions of Enum·Union, on the input $\{\{1,2\},\{2,3\},\{1,4\},\{1,3,4\}\}$. The branches containing no solution are not explored by the algorithm nor represented in the figure.

The second method to solve an enumeration problem ENUM·A in polynomial delay is to define a graph with set of vertices $A(x)$, often called the *supergraph*. Assume that the supergraph satisfies the following properties:

- The supergraph is connected.

- The neighborhood of a vertex can be produced in polynomial time.

Then, a depth first traversal of the graph allows to find all solutions with polynomial delay. Often, the supegraph is oriented, then to obtain a polynomial delay traversal, either it should be strongly connected or we must be able to efficiently compute a set of vertices from which all other vertices can be reached. Several methods have been proposed to define strongly connected supergraphs, such as proximity search [Conte & Uno 2019] or retaliation-free paths [Cao 2020].

Contrarily to the flashlight method, the supergraph method requires as much space as there are solutions, since all elements of $A(x)$ must be marked (and thus kept in memory) to realize the traversal. To get rid of the space, one can try to define a spanning tree or a spanning forest over the supergraph. If the children and the parent of any vertex in the spanning tree can be found in polynomial time, then a traversal of the spanning tree is an enumeration algorithm and it does not need to store all solutions but only a single one. This method, called the *reverse search*, has been introduced by Avis and Fukuda [Avis & Fukuda 1996] to solve problems like enumerating triangulations of points in the plane, spanning trees of a graph or topological orderings of an acyclic graph.

Example 6. The supergraph method can be used to solve ENUM·UNION. Let $x = \{s_1, \ldots, s_m\}$ an instance of ENUM·UNION, with for all i, $s_i \subseteq [n]$. We define a directed graph G_x with for vertices, the elements of UNION(x) and $\emptyset$. There is an arc (y_1, y_2) in G_x if there exists i such that $y_2 = y_1 \cup s_i$. Clearly, any element in UNION(x) can be reached from $\emptyset$, hence traversing G_x solves ENUM·UNION with polynomial delay. See Figure 3.5 for an example.

An implicit spaning tree of the supergraph can be defined as follows: y_1 is the parent of y_2 if it is the smallest lexicographic element such that (y_1, y_2) is an arc of the supergraph. When traversing G_x, it is possible to skip all arcs which are not in the parent relation, by computing the smallest set in $A(x)$ which can yield some y by union with an element of x. This can be done in polynomial time, but with higher complexity than the binary partition method presented in the previous example.

3.4   Separation of Enumeration Classes

In the previous sections, we have provided several separations, under various complexity hypotheses, which can be summarized by the following chain of inclusions:

$$\text{DelayP} = \text{IncP}_1 \subsetneq \cdots \subsetneq \text{IncP}_i \subsetneq \cdots \subset \text{IncP} \subsetneq \text{OutputP} \subsetneq \text{EnumP}.$$

As explained in [Capelli & Strozecki 2019], these separations are unconditional if we remove the constraints that problems are in EnumP, i.e. that the solutions can be verified in
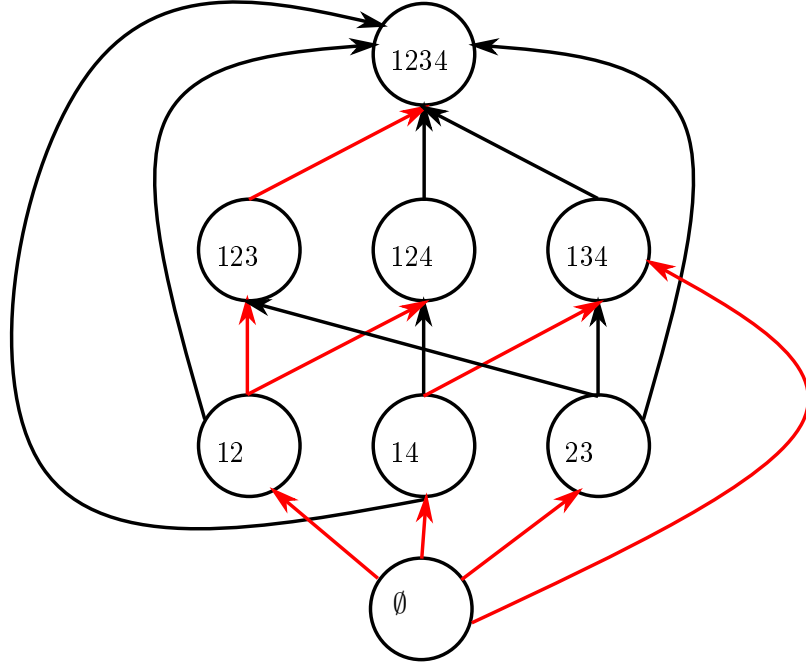
Figure 3.5: The supergraph of solutions of Enum·Union, on the input $\{\{1,2\},\{2,3\},\{1,4\},\{1,3,4\}\}$. Arcs of the implicit spanning tree in red.

polynomial time, by using the time hierarchy Theorem [Hartmanis & Stearns 1965]. However, several properties do not hold in that case, such as the closure of DelayP under union presented in the next section or the equivalences between separation of enumeration complexity classes and $\mathsf{P} \neq \mathsf{NP}$ or $\mathsf{FP} \neq \mathsf{TFNP}$.

Consider the SAT problem, then ExtSol·SAT is equivalent to SAT, that is SAT is autoreducible. Hence, if $\mathsf{P} = \mathsf{NP}$, then ExtSol·SAT $\in P$ and using the binary partition method, Enum·SAT $\in$ DelayP, which yields the following proposition.

Proposition 3.6. $\mathsf{P} = \mathsf{NP} \Leftrightarrow \text{EnumP} = \text{DelayP}$.

The last proposition is somehow surprising, since it implies that separating EnumP from DelayP is as hard as separating it from OutputP.

3.5 Stability under Operations and Reductions

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 it defines the new problem Enum·$(A \cup B)$. This definition can easily be adapted to any set operation such as the complement or the intersection. We can also define the deletion or subtraction of B from A as $(A \setminus B)(x) = A(x) \setminus B(x)$, with the additional constraint that $B(x) \subseteq A(x)$. This is inspired by the subtractive reduction, defined for counting problems [Durand *et al.* 2005]. If $B(x)$ is polynomial in $|x|$, then we say that $(A \setminus B)$ is a polynomial size deletion.

We say that a complexity class $\mathcal{C}$ is *stable* under some operation, say the union, when ENUM·$A \in \mathcal{C}$ and ENUM·$B \in \mathcal{C}$ implies ENUM·$A \cup B \in \mathcal{C}$. It turns out that DelayP is stable for several operations.

Theorem 3.2 ([Bagan 2009],[Strozecki 2010b]). *The classes* OutputP, IncP, DelayP *are stable under union, Cartesian product and polynomial size deletion.*

The proof for cartesian product or polynomial size deletion are straightforward. The method for solving $A \cup B$ relies on a simple priority scheme: compute the next solution of A, if it is not a solution of B then output it. If it is a solution of B, then output the next solution of B. This allows to deal with repetition of solutions and only multiply the delay by two, hence polynomial delay is ensured for a union of a polynomial number of problems in DelayP. If the solutions of A and B are generated in the same order, then it is not necessary to test whether a solution of A is also a solution of B, by dynamically merging the two ordered streams of solutions.

As a consequence, these operations can be used to decompose enumerations problems into simpler problems and thus may simplify the design of good algorithms.

Several other operations do not leave any enumeration classes stable but EnumP because they allow to build an EnumP-hard problem from simpler ones. For instance, deciding whether there is a model to the conjunction of a Horn Formula and an affine formula is NP-complete [Creignou *et al.* 2001], while enumerating the models of a Horn or an affine formula can be done with polynomial delay [Creignou & Hébrard 1997].

Theorem 3.3 ([Strozecki 2010b]). *If* P $\neq$ NP *then* DelayP, IncP, OutputP *are not stable under deletion, intersection or complement.*

Inspired by these operations, we would like to design the most general parsimonious reduction, so that DelayP is stable under its action. As explained in the previous chapter, to make ENUM·$3SAT_0$ complete, it is enough to generalize parsimonious reduction by allowing addition or deletion of a polynomial number of solutions themselves produced in polynomial time. DelayP is stable under such reductions because of Proposition 3.2. Several reductions for OutputP, IncP and DelayP have been given by Mary in [Mary 2013]. 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 8 (Polynomial delay reduction [Mary 2013]). *Let* ENUM·A, ENUM·$B \in$ EnumP *and* f, g *two polynomial time computable functions. The function* f *maps instances of* ENUM·A *to instances of* ENUM·B, *while* g *maps a solution of* ENUM·B *to a set of solutions of* ENUM·A. *The pair* (f, g) *is a polynomial delay reduction from* ENUM·A *to* ENUM·B *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 listing minimal transversals of a hypergraph is equivalent to listing the minimal dominating sets of a graph [Kanté *et al.* 2014], a result which has motivated the study of enumeration of minimal dominating sets over various graph classes [Kanté *et al.* 2015, Bonamy *et al.* 2019].

We can relax polynomial delay reductions by allowing a polynomial number of repetitions, such reductions are called e-reduction in [Creignou *et al.* 2017a]. 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.

To define a Turing reduction for enumeration, we give access to an oracle solving an enumeration problem using two additional instructions, one to begin an enumeration, the other to get the next solution. A reduction from A to B is a machine which solves ENUM·A using an oracle to the problem ENUM·B. When the machine has a polynomial delay or an incremental polynomial time, the reductions are called D or I reductions [Creignou *et al.* 2017a].

No complete problems are known for the complexity classes we have introduced but EnumP, whatever the reduction considered. In fact, artificial constructions using padding, as presented for 3SAT and parsimonious reduction, are often enough to build a hard but not complete problem. Moreover, when IncP$_i$ is stable under a reduction, which is the case for all mentioned reductions but the Turing reductions, there is no complete problem in IncP because it implies a collapse of the strict hierarchy given in Theorem 3.1.

Corollary 3.1. *Let $\prec$ be a reduction for enumeration problems such that, for all $a \geq 1$, IncP$_a$ is stable under $\prec$. If* ETH *holds, there is no complete problem in* IncP *for $\prec$.*

This lack of complete problems underlines the need to find other methods to prove hardness of enumeration problems.

Space Complexity

Contents

In the previous Chapter, only time restrictions have been considered, but space plays a major role in practical enumeration algorithms and is often a bottleneck, for instance when generating maximal cliques [Conte *et al.* 2016]. We present several complexity classes, designed to capture the idea of bounded space and study their relations with their counterparts with unbounded space.

4.1 Polynomial Space

We could define an equivalent of PSPACE by considering all enumeration problems which can be solved by a machine which uses a space polynomial in the input size. Such a class contains EnumP, and all the polynomial hierarchy defined in [Creignou *et al.* 2017a].

As a tractability measure, it is more relevant to ask for algorithms with polynomial space and polynomial delay. In fact, several algorithmic methods have been designed, to realize the traversal of a supergraph of solutions with polynomial delay and *polynomial space* [Lawler *et al.* 1980, Avis & Fukuda 1996, Conte *et al.* 2019]. We denote by DelayP^{poly} the class of problems solvable in polynomial delay and polynomial space, and by IncP^{poly} the class of problems solvable in incremental polynomial time and polynomial space. 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 ENUM·MIN-TRANSVERSALS $\notin$ DelayP^{poly}, which should be easier than proving it is not in DelayP.

Several properties of IncP do not hold for IncP^{poly}. The amortization method which allows to trade space for regularity as in Proposition 3.3 does not work when requiring polynomial space. Hence, with polynomial space IncP$_1^{poly}$ may not be equal to DelayP^{poly}. We deal with this problem at the end of the chapter. Moreover, it does not seem possible to characterize

the class IncP^{poly} by a variant of the problem AnotherSol, which makes it quite different from IncP.

A natural question is to understand whether the use of exponential space is useful for DelayP or IncP. If we relax the constraint that the problems must be in EnumP, then we can define a problem with an exponential number of trivial polynomial size solutions and one which is the solution to an EXP-complete problem. Such a problem is in DelayP, since outputting the trivial solutions gives enough time to solve the EXP-complete problem. However, if this problem is in DelayP^{poly}, then the enumeration algorithm is also a polynomial space algorithm solving an EXP-complete problem, which implies the unlikely collapse PSPACE = EXP.

Open problem 4.1. *Prove that* DelayP $\neq$ DelayP^{poly} *and* IncP $\neq$ IncP^{poly} *assuming some complexity hypothesis.*

4.2 Memoryless Polynomial Delay

A notion of memoryless polynomial delay algorithm is proposed in [Strozecki 2010b], under the name of strong polynomial delay, which is used in this thesis for another concept. A memoryless algorithm can use only the last solution and the input to produce the next solution.

We say that a problem ENUM·A is in NextP if for every instance x there is a total order $<_x$ such that the following problems are in FP:

1. given x, output the first element of $A(x)$ for $<_x$

2. given x and $y \in A(x)$ output the next element of $A(x)$ for $<_x$ or a special value if there is none

The enumeration of the minimal spanning trees or of the maximal matchings of a weighted graph are in NextP (see [Avis & Fukuda 1996, Uno 1997]). Many problems of queries over databases are solved by implementing a Next function, which given a solution produces the next one (often in constant time), and a Start function which gives the first solution, see for instance [Schweikardt *et al.* 2018]. In fact, any problem which can be solved by traversal of an implicit tree of solutions, without storing global information, is in NextP. As a consequence, reverse search algorithms or flashlight algorithms are often memoryless.

Several results proved for DelayP also hold for NextP. If P = NP, then EnumP = DelayP^{poly} = NextP, since ENUM·SAT can be solved using a flashlight search. Moreover, NextP is stable under cartesian product, disjoint union and polynomial size deletion, using the same methods as for DelayP, but the method for proving stability under union is not memoryless.

Open problem 4.2. *Can we separate* NextP *from* DelayP *modulo some complexity hypotesis ? If so, can we separate* NextP *from* DelayP^{poly}*, by taking advantage that a* DelayP^{poly} *algorithm can store an information computed during the whole enumeration process, while a* NextP *algorithm uses a "local" information to find the next solution.*

4.3   Sources of Exponential Space

There are three common sources of exponential space in enumeration.

The most common is that many algorithms generate each solution several times, a typical example is the algorithm used to generate maximal independent sets in lexicographic order given in [Johnson *et al.* 1988]. Any algorithm can be turned into an enumeration algorithm without repetition by storing all output solutions in a trie to detect and avoid repetitions. If the number of repetitions is bounded by some function $f(n)$, with n the size of the instance, then the elimination of duplicates multiplies the incremental time of the algorithm with repetition by $f(n)$ only. However, this requires to store all output solutions, which may exponentially increase the space used.

When an algorithm produces some solution s at time t, to decide whether s has already been output, we can run the enumeration from start to time t. This method requires no *additional space*. If the original algorithm is in incremental time $k^i p(n)$, and the number of repetitions of a solution is bounded by $f(n)$, then the algorithm without repetition is in incremental time $O(k^{i+1} p(n) f(n))$. Hence, a problem which is in IncP_i because of an algorithm using polynomial space and an exponential datastructure to detect duplicates in polynomial number, is also in IncP_{i+1}^{poly} using this method. There is a tradeoff between incremental time and space using a hybrid of these two methods to avoid repetitions: store the last l produced solutions and rerun the algorithm to test whether a solution is output only up to the point the first of the l stored solutions is produced.

The second source of exponential space is the use of saturation algorithms: Since they compute new solutions from old ones, they require to store all produced solutions. There is no known generic method to get rid of the space in that case. However, it may be possible to design a different algorithm to solve the problem, see Chapter 7 where we show that a whole class of saturation problems can be solved with polynomial space and polynomial delay using a flashlight search.

Open problem 4.3. *The enumeration of the circuits of a binary matroid can be done in incremental polynomial time, using a saturation algorithm [Khachiyan et al. 2005]. Can we achieve the same complexity with a polynomial space only?*

The third common source of exponential space is the amortization procedure of Proposition 3.3. It builds a polynomial delay algorithm from one in linear incremental time, by storing the produced solutions in a queue, which can be as large as the whole set of solutions. We show in the next section, that this source of exponential space can be eliminated.

4.4   Amortization in Polynomial Space

Let M be a machine in linear incremental time, it produces k solutions in time $kp(n)$. We say that $p(n)$ is the *incremental delay* of M. The aim of this section is to show how to build a machine M', which produces the same solutions as M, with a polynomial delay almost equal to the incremental delay of M and with a small space overhead.

4.4.1   Gaps

In [Capelli & Strozecki 2019], we show how to amortize an incremental linear time enumeration, if it is regular enough. To capture the notion of regularity, we define the notion of gap for a machine M: i is a *p-gap* of M (for x) if $T(x, i+1) - T(x, i) > p(|x|)$. If there is a polynomial p, such that M has no p-gap, then M has delay p. In fact, if M has only few p-gaps, it can be turned into a polynomial delay algorithm, see [Capelli & Strozecki 2019].

We can prove something more general, by requiring the existence of a large interval of solutions without p-gaps rather than bounding the number of gaps, as stated in Proposition 4.1. It captures more cases, for instance an algorithm which outputs an exponential number of solutions at the beginning without gaps and then has a superpolynomial number of gaps. The idea is to compensate for the gaps by using the dense parts of the enumeration. It is done by running several copies of the incremental linear algorithm, and detecting intervals of time in the eumeration with many output solutions, so that it can be used to amortize intervals with fewer solutions. The proof is quite technical, see [Capelli & Strozecki 2019].

Proposition 4.1. *Let M be an incremental linear time machine solving* ENUM·A *using polynomial space. Assume there are two polynomials p and q such that for all x of size n, and for all $k \leq |A(x)|$ there exists $a < b \leq k$ such that $b - a > \frac{k}{q(n)}$ and there are no p-gaps between the ath and the bth solution. Then* ENUM·$A \in \mathrm{DelayP}^{poly}$.

4.4.2   Geometric Amortization

The method of the last section is not sufficient to prove $\mathrm{IncP}_1^{poly} = \mathrm{DelayP}^{poly}$. To do so, we introduce a new amortization method, which is also based on simulating the incremental linear algorithm at several points in time. It is simpler, more general and more efficient than the one designed to prove Proposition 4.1. We give here the algorithm and a sketch of its analysis; for more details and related results, see [Capelli & Strozecki 2021b].

To simplify the presentation, we consider that the computation of the incremental linear machine M on an input x can be represented by a list $L_{M,x}$. If M outputs a solution at time step i, then the ith element of $L_{M,x}$ contains this solution, otherwise it contains a special value denoting the absence of solution. It is possible to go from an element of $L_{M,x}$ to the next one by simulating one computation step of M. To store a pointer to the ith element of $L_{M,x}$, we store the state of M at time step i.

We describe the complexity of an algorithm working on such a list as the number of reads of elements of $L_{M,x}$ for the time and the number of stored pointers for the space. In particular, for an algorithm outputting solutions of $L_{M,x}$, the delay of the algorithm is the maximal number of reads between the output of two distinct solutions or between the output of a solution and the end of the algorithm. We ignore the cost of maintaining counters, of switching between different contexts of computation etc. It can be shown that these costs amount only to a constant time overhead on a RAM machine [Capelli & Strozecki 2021b].

The pseudo code of the *geometric amortization* method we propose is given in Algorithm 1. Let M be the machine we amortize, it is in incremental delay $p(n)$ and outputs ℓ solutions on input x. We assume that $p(n)$ and ℓ, or a bound on their values, are known. The idea is to maintain a list p of $N := 1 + \lceil \log(\ell) \rceil$ pointers to elements of $L_{M,x}$. At the start of the

algorithm, we initialize each $p[i]$ for $i < N$ to be a pointer to the first entry of $L_{M,x}$. We also initialize a table c of N counters, whose entries are initialized to 1. Each time $p[i]$ is moved to the next entry of $L_{M,x}$, we increment $c[i]$ so that $p[i]$ always points toward $L[c[i]]$ (lines 9-10 in Algorithm 1).

Each $p[i]$ explores $L_{M,x}$. If at some point in the algorithm, $p[i]$ points toward a solution of $L_{M,x}$, and if $2^{i-1}p(n) < c[i] \leq 2^i p(n)$, then we output the solution (for $i = 0$, we take the bounds $1 \leq c[0] \leq p(n)$). Thus, solutions will be output at most once since the index of the solution determines which pointer discovers it.

We now describe the mechanism we use to move the different pointers. We move one pointer $p[j]$ for at most $2p(n)$ steps (corresponding to the loop starting at Line 7 in Algorithm 1). If we find a solution in the zone corresponding to $p[j]$, that is $[1 + 2^{j-1}p(n); 2^j p(n)]$, we output it and move back to pointer $p[N-1]$. If we move by more than $2p(n)$ steps without finding a solution in the right zone, we decrease j by 1 and continue the exploration. If j is 0 and no solution has been found in the zone of $p[0]$, that is $[1; p(n)]$, we stop the algorithm.

Algorithm 1: Pseudocode for the geometric amortization method

    Data: A list L containing ℓ solutions and an incremental delay $p(n)$

1 **begin**
2     $N \leftarrow 1 + \lceil \log(\ell) \rceil$;
3   Initialize a table p of size N containing N pointers to $L[1]$;
4   Initialize a table c of size N containing N integers set to 1;
5     $j \leftarrow N - 1$;
6   **while** $j > 0$ **do**
7   **for** $b \leftarrow 2p(n)$ **to** 0 **do**
8           $s \leftarrow$ the value of the entry of L pointed by $p[j]$;
9   Move $p[j]$ to the right;
10         $c[j] \leftarrow c[j] + 1$;
11   **if** s *is a solution* **and** $1 + 2^{j-1}p(n) \leq c[j] \leq 2^j p(n)$ **then**
12   Output solution s;
13             $j \leftarrow N - 1$;
14   **break**;
15   **if** $b = 0$ **then**
16           $j \leftarrow j - 1$

Theorem 4.1 ([Capelli & Strozecki 2021b]). *Let M be a machine in incremental delay $p(n)$, space $s(n)$. Algorithm 1 on input $(L_{M,x}, \ell, p(|x|))$ outputs all solutions of $L_{M,x}$ with a delay $O(\log(\ell)p(n))$, space $O(\log(\ell)(s(n) + \log(\ell)))$.*

Proof. (Rough sketch) Algorithm 1 uses $O(\log(\ell))$ pointers in the table p. Each of them can be stored with space in $O(S)$ where S is the space used by the machine M. It is easy to see by construction that the delay between two solutions is $O(\log(\ell)p(n))$. Indeed, the complexity of executing the loop starting at Line 7 is $O(p(n))$. Now, either a solution is found before

Figure 4.1: The interface of Coussinet which takes an IncP$_1$ list of outputs and does the amortization as in Theorem 4.1.

j reaches 0, in which case, we have a delay of at most $O(Np(n)) = O(\log(\ell)p(n))$ since j is decreasing and starts at $N - 1$. Or j reaches 0 and no solution is found and this is the end of the algorithm, thus there is a delay of at most $O(\log(\ell)p(n))$ between the last output solution and the end of the algorithm.

As an invariant of the algorithm, we prove that $c[i + 1]$, that is the number of steps done by the $i + 1$st pointers is larger than the number of solutions produced by the first i pointers times $p(n)$. It can then be used to prove that Algorithm 1 does not stop before outputing all solutions. □

Let us consider a slight variation of Algorithm 1, in which a pointer which has finished to explore its zone is removed from the list of pointers. In that case, on average, each entry of $L_{M,x}$ is read at most twice, which implies that the total time of the geometric amortization is the same as the original enumeration, up to a constant factor.

An implementation of the geometric amortization method called Coussinet is available on Florent Capelli's website, see Figure 4.1 for an illustration.

As a corollary of Theorem 4.1, we obtain a collapse of complexity classes, which was an open question in [Capelli & Strozecki 2019].

Corollary 4.1. IncP$_1^{poly}$ = DelayP^{poly}.

As a consequence of this corollary, the right measure of tractability seems to be incremental linear time rather than polynomial delay. Indeed, it is not clear that polynomial delay is really necessary in applications and we have proved that if it is, we can obtain it from a linear incremental algorithm, with a small space overhead and no change in the total time. As an application of Theorem 4.1, we amortize a flashlight search solving ENUM·DNF to turn an average delay into a delay in Chapter 5.

4.4.3 Amortizing without Assumptions

To amortize a machine M on an input x thanks to Algorithm 1, we must know these values (or a bound on them):

- the space used by the machine M on x

- the number of solutions produced by M on x

- the incremental delay of M on x

We briefly explain how we can do without knowing the first two values, while the third one is more problematic (see [Capelli & Strozecki 2021b]).

The space used by M is used implicitly in the algorithm to store the pointers to different parts of the list $L_{M,x}$. Indeed, a pointer represents the state of the machine at some point of its computation, hence we need to store all used registers. If we have a bound on this space, we reserve enough space for each pointer at the beginning of the algorithm. If we do not know how much space is used, we can use a dynamic array, with lazy initialization and expansion. The overhead is either logarithmic in time or n^ε in space, for any ε, depending on the expansion factor of the dynamic array.

The number of solutions in $L_{M,x}$ is an input of Algorithm 1, it is used to determine the number of pointers. Note that, in Algorithm 1, all pointers which are not yet in their zone move together, one after the other. Hence, we could just move one of them to represent them all. In this variant, all pointers are in their zone, except the last one. When the last one enters its zone, we can duplicate it to create a new pointer for the next zone. This algorithm mimicks Algorithm 1, but does not require the number of pointers beforehand. If the machine M does not use more than $O(t)$ space after t time steps, then the copy operation to obtain a new pointer is done lazily with a constant time overhead. If we have no guarantee on M, the time overhead is at most logarithmic.

Finally, we need the incremental delay of the machine M, to move each pointer in $L_{M,x}$ by twice this number of steps before going to the previous pointer. In the classical amortization method using a queue, the incremental delay is also required, since it controls when an element is pulled out of the queue to be output. We prove that the classical amortization algorithm can be made to work, even when the incremental delay is unknown. The idea is to maintain a bound on the incremental delay of a machine M, while simulating it. We use a value larger than this bound, which depends on a fixed parameter ε, to output elements out of the queue. This gives a polynomial delay algorithm, but the obtained delay is worse than the incremental delay of the original algorithm.

Proposition 4.2 ([Capelli & Strozecki 2021b]). *For any $\varepsilon > 0$, and machine M in incremental delay $O(p(n))$, there is a machine M_ε which enumerates all outputs of M in the same order, in delay $O(p(n)^{1+\varepsilon})$.*

Open problem 4.4. *Can we use the method of dynamically evaluating the incremental delay in Algorithm 1? Prove that there is a machine, which takes as input another machine in incremental linear time and polynomial space and enumerates the same solutions in polynomial delay and polynomial space.*

When amortizing without knowing the incremental delay, it seems impossible to obtain a delay equal to the incremental delay. To prove a lower bound, we assume that the amortization method can only work on $L_{M,x}$ and not on the machine M and input x directly. To further simplify, in the complexity we only take into account a read in L_M as an elementary operation, and we allow an unbounded memory. Since the memory is unbounded, every solution found while exploring $L_{M,x}$ can be stored in a queue to be later output, hence an amortization method only needs to go through $L_{M,x}$ once. In these settings, an efficient amortization method explores the list $L_{M,x}$, stores the solutions in a queue and its only freedom is to determine when solutions from the queue should be output. This depends only on

the number of elements of $L_{M,x}$ seen, the number of solutions encountered and the time since the last output. When such an amortization algorithm outputs solutions from the queue too fast, it is possible to build an adversary list and prove the following theorem.

Proposition 4.3 ([Capelli & Strozecki 2021b]). *There is no RAM machine, which given a list $L_{M,x}$ representing the computation of a machine M on x in incremental delay $p(|x|)$, enumerates all outputs of M in delay $(s-2)p(|x|)$ where s is the number of solutions generated by M.*

Below Polynomial Delay

Contents

5.1   Strong Polynomial Delay

When the input is huge with regard to a single solution, we would like a sublinear delay in the input size, or even a delay which depends only on the size of a single generated solution. This situation naturally appears when the input is a hypergraph and solutions are subsets of its vertices, or when solutions are of constant size, for instance when enumerating solutions of a first order query over a database. 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 9 (Strong Polynomial delay). *A problem* ENUM·$A \in$ EnumP *is in* SDelayP *if there exist constants $a \geq 0$ and $C > 0$, and there is a machine M which solves* ENUM·A *for all inputs x such that:*

- *$T(x, 1) \leq p(|x|)$, with p a polynomial (polynomial preprocessing)*

- *for all $1 < k \leq |A(x)|$, $|T(x, k) - T(x, k - 1)| < C|y_k|^a$, where y_k is the kth solution output by M (strong polynomial delay).*

Presently, very few problems have strong polynomial delay algorithms: generating $s - t$ paths in a DAG, generating assignments of existential FO formulas with second order free variables [Durand & Strozecki 2011] or existential MSO formulas over bounded tree width structures [Courcelle 2009, Amarilli *et al.* 2017]. When solving an enumeration problem with an infinite set of solutions [Florêncio *et al.* 2015], any reference to the input size or the total time is irrelevant and strong polynomial delay is the proper notion of tractability. Moreover, a strong polynomial delay algorithm can be applied to large inputs given implicitly: this idea is used to give polynomial delay algorithms for generating dominating sets over several classes of graphs [Blind *et al.* 2020].

Even an extremely simple problem like ENUM·DNF, the enumeration of models (or satisfying assignments) of a DNF formula, has no simple algorithm in strong polynomial delay. A formula in disjunctive normal form (DNF) is a disjunction of terms. Each term

is a conjunction of litterals, that is variables or negation of variables. For instance $D = (X_1 \wedge \neg X_2) \vee (X_2 \wedge X_3)$ is a DNF formula with two terms over the variables $\{X_1, X_2, X_3\}$, and four models $\{(1, 0, 0), (1, 0, 1), (0, 1, 1), (1, 1, 1)\}$. The structure of the assignments of a DNF is extremely regular: it is the *union* of the models of its terms. A term sets the values of some variables, and we can enumerate all possible values of the remaining variables in constant delay using Gray Code. The main difficulty to obtain an algorithm with strong polynomial delay for ENUM·DNF is that *the union is not disjoint* and the simple algorithm consisting in enumerating the solutions of each term one after the other produces redundant solutions. Solution repetitions because of non disjoint union is a common problem in enumeration and this issue appears in its simplest form when solving ENUM·DNF. We hope that understanding finely the complexity of this problem and designing better algorithms to solve it will shed some light on other similar problems. We conjecture that ENUM·DNF cannot be solved with strong polynomial delay.

Conjecture 5.1.1 (DNF Enumeration Conjecture). ENUM·DNF $\notin$ SDelayP.

We did even state a Strong DNF Enumeration Conjecture in [Capelli & Strozecki 2021a], similar in precision to ETH, stating that no algorithm solves the problem ENUM·DNF in delay $o(m)$, where m is the number of terms of the DNF. We show later in this section that this conjecture does not hold.

To solve ENUM·DNF, a flashlight search can be used. Indeed, EXTSOL·DNF can be solved in time $|D|$, where D is the input DNF. In fact, the complexity of solving the extension problem can be amortized over a branch of the tree of partial solutions, so that the delay of the enumeration algorithm is $|D|$ [Capelli & Strozecki 2021a]. It turns out that the flashlight method can be improved in several directions. First, the input DNF is stored as a trie [Fredkin 1960], and when going down the tree of partial solutions, variables are fixed and the size of the trie is decreased. Second, we proved that a DNF with m terms has at least $m^{\log_3(2)}$ models, then we use that fact to prove a good bound on the average delay as in Uno's push-out method [Uno 2015].

Let n be the number of variables of D. If the trie is shrunk very carefully when fixing variables, a factor n can be shaved from the delay. We can further improve the algorithm and its analysis for the case of monotone DNFs, by further compacting the represented DNF. For k-DNF formulas, i.e. DNF formulas with terms of size at most k, we use a particular traversal of the tree of partial solutions. It allows to find a k-term, whose solutions are disjoint from the ones in the other branches of the tree. These solutions are output with constant delay (or implicitly) and can be used to amortize the time of going down the tree of partial solutions up to the point there are only $2k$ variables left. This yields a *constant delay* algorithm for fixed k. The results of [Capelli & Strozecki 2021a] are given in Table 5.1.

We have obtained algorithms with good *average delay*. To falsify the Strong DNF Enumeration Conjecture and the DNF Enumeration Conjecture on the class of monotone formulas, we need to replace the average delay by a similar delay. To do that, we use a generic method [Capelli & Strozecki 2021b] to transform an average delay into a real delay in flashlight algorithms, relying on the amortization method of Theorem 4.1.

Let the **path time** of a flashlight search algorithm be the time needed by a flashlight search algorithm to go from the root of the tree of partial solutions to a leaf. For most

Class	Delay	Space
DNF	$O(\|D\|)$	Polynomial
($\star$) DNF	$O(nm^{1-\log_3(2)})$ average delay	Polynomial
($\star$) k-DNF	$2^{O(k)}$	Polynomial
($\star$) Monotone DNF	$O(n^2)$	Exponential
($\star$) Monotone DNF	$O(\log(nm))$ average delay	Polynomial

Table 5.1: In this table, D is a DNF, n its number of variables and m its number of terms.

flashlight algorithms, path time and delay are the same. The main idea is that, at any point of a flashlight algorithm, we are exploring some vertex of the tree of partial solutions. The vertices which have already been explored are whole subtrees, corresponding to subproblems of the enumeration problem being solved, plus a path from the root, as shown in Figure 5.1. This allows to precisely bound the incremental time of flashlight search. In particular, it is linear (up to the path time), if the average delay is polynomial, because the time spent in subproblems is linear in the number of output solutions. If the path time is bounded by $p(n)$, the enumeration of solutions can be delayed by $p(n)$, using a queue of size at most $p(n)$, which gives an algorithm in incremental linear time. Then we can apply Theorem 4.1 to make the delay polynomial and we obtain the following theorem.

Theorem 5.1 ([Capelli & Strozecki 2021b]). *Let* Enum$\cdot$A *be an enumeration problem, solved by a flashlight search algorithm, in polynomial space, polynomial path time and average delay* $a(n, k)$, *where* n *is the size of the instance and* k *a bound on the size of a solution. Let* N *be the number of produced solutions, then there is an algorithm solving* Enum$\cdot$A, *with delay* $a(n, k) \log(N)$, *average delay* $a(n, k)$ *and polynomial space.*

Open problem 5.1. *We are left with a single unsettled conjecture, the DNF Enumeration Conjecture. Prove it assuming some complexity hypothesis like* ETH. *As a first step, it should be proven that* Enum$\cdot$DNF *cannot be solved in constant delay or logarithmic delay.*

The problem Enum$\cdot$Union of computing the closure by union of a set system has a linear delay algorithm [Mary & Strozecki 2019]. Because a solution can be obtained by many different unions, Enum$\cdot$Union does not seem to have a strong polynomial delay algorithm. In [Capelli & Strozecki 2021a], we **wrongly** claim that a method similar to the one presented for Enum$\cdot$DNF on monotone formulas solves Enum$\cdot$Union with average delay polynomial in the solution size. Indeed, the problem solved when traversing the tree of partial solutions is *more complex* than the original problem, because some elements are forced to be in the solution. This more complex problem can be solved in linear time, but it is not possible to prove a lower bound on its number of solutions with regards to the input size.

Open problem 5.2. *Give an algorithm for* Enum$\cdot$Union *with average delay in* $o(m)$, *where* m *is the number of sets in the input.*

Instead of focusing on specific problems to understand the difference between SDelayP and DelayP, we could just prove a separation result using an artificial problem. However,
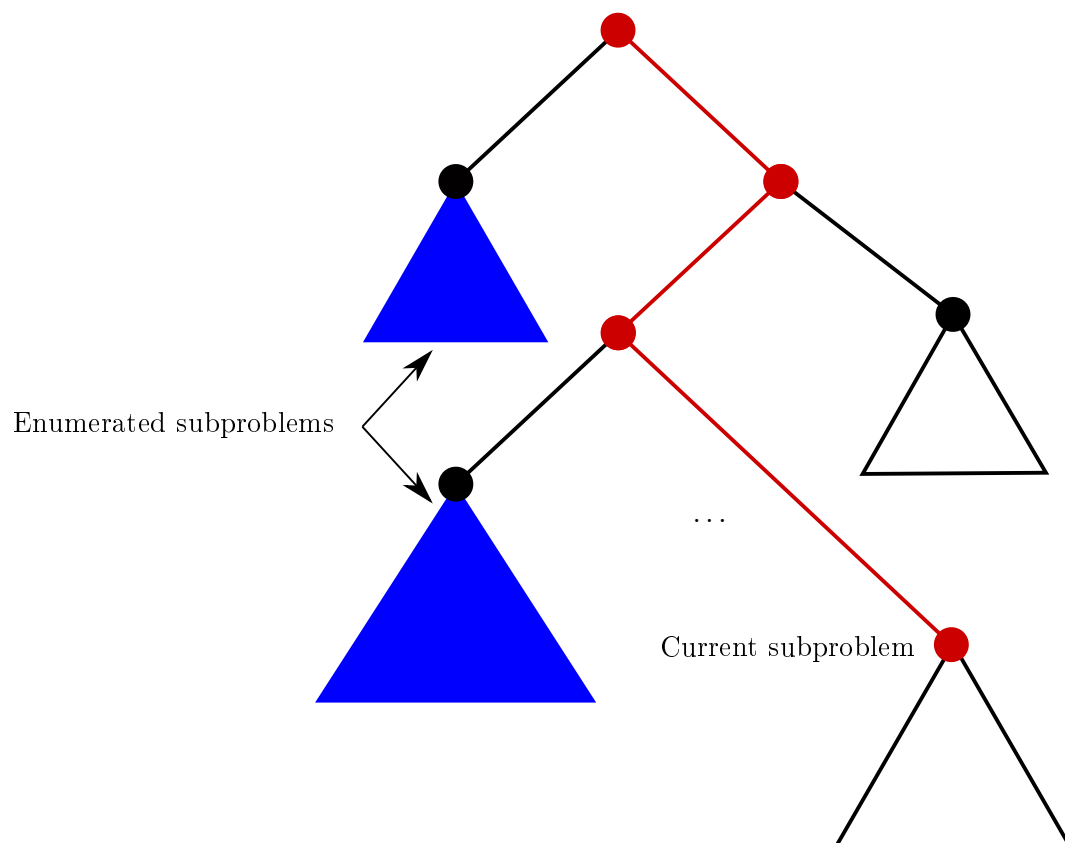
Figure 5.1: A traversal of the tree of partial solutions by the flashlight search. The subproblems completely solved recursively in blue, the path to the current partial solution in red and the unexplored vertices in the white triangles.

the method to prove a hierarchy inside IncP does not seem useful to prove the same kind of hierarchy inside DelayP.

Open problem 5.3. *Prove that* SDelayP $\subsetneq$ DelayP *modulo a complexity hypothesis like* ETH.

If we drop the condition that problems must be in EnumP, the previous question becomes easy, as for the other class separations presented in Chapter 3.

Proposition 5.1. *There is an enumeration problem with an algorithm in polynomial delay and polynomial preprocessing but no algorithm in strong polynomial delay and polynomial preprocessing.*

Proof. Using the time hierarchy theorem [Hartmanis & Stearns 1965], we obtain a language L which can be solved in time $O(n2^{\log(n)^2})$ but not in time $o(n2^{\log(n)^2}/\log(n)^2)$. We let $n = |x|$ and we let A be the binary predicate defined as $A(x) = [2^{\log(n)^2}]$ if $x \notin L$ and $A(x) = [2^{\log(n)^2}] \cup \{\sharp\}$ otherwise. The set $A(x)$ consists in $2^{\log(n)^2}$ trivial solutions and $\sharp$ when x is in L.

Assume now that we can enumerate the solutions of ENUM·A in strong polynomial delay. Since the solutions are of size $\log(n)^2$, the total time to generate all solutions is less than $O(2^{\log(n)^2}p(\log(n)))$, with p a polynomial. The polynomial time preprocessing does not appears in the total time since it is dominated by the generation time. Solving ENUM·A allows to decide L, which cannot be done in time $O(2^{\log(n)^2}p(\log(n)))$, hence ENUM·$A$ cannot be solved in strong polynomial delay.

However, ENUM·A can be solved in linear delay. Enumerate all trivial solutions in $A(x)$ with delay n. Since there are $O(2^{\log(n)^2})$ solutions, the total time to generate these solutions is in $O(n2^{\log(n)^2})$, hence we can use this time to decide whether $x \in L$ and to produce the last solution. $\square$

5.2 Constant Delay

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

Gray code In several RAM models, the output registers are fixed and an output solution is always contained in the first output registers. Hence, two consecutive solutions must only differ by a fixed number of registers, such an ordering of solutions is often called a Gray code. If the solution is represented by 0 and 1 in the output registers, it means that the Hamming distance between consecutive solutions is constant. In our computation model, the registers may contain integers less than n that can be manipulated in constant time, hence the Hamming distance between two consecutive solutions is bounded by $O(\log(n))$. Since the algorithm goes from a solution to the next in constant time, it is sometimes called a loopless algorithm in the context of Gray codes. There are many such algorithms for the generation

of combinatorial objects [Walsh 2003, Ruskey 2003, Knuth 2011, Hartung *et al.* 2019]: the integers less than 2^n, combinations, Dyck words, . . .

Example 7. Let RBC_n be the reflected binary code order on n bits words, represented as a list. It is defined recursively as the concatenation of the words of RBC_{n-1} prefixed by 0 followed by the words of RBC_{n-1} in reverse order prefixed by one. To go from one word to the next in RBC_n, the 1 in the current word should be maintained as a list as well as the parity of the number of ones. If the number of ones is even, the last bit is flipped, otherwise the bit to the left of the rightmost one is flipped. Since the word is represented as an array of size n, all these operations can be done in constant time.

Constant delay with dynamic amortization   To generate very different consecutive solutions in constant delay, we must relax the previous definition. For instance, it seems natural when several solutions are already computed in memory, to be able to output them in constant delay. This is why we have defined RAM machines for enumeration with a generalized OUTPUT instruction, which can output any region of the memory. It allows to output already computed solutions in constant delay or to amortize a slow enumeration process with a fast one. It can be used to generate models of a k-DNF in constant delay [Capelli & Strozecki 2021a]. This form of constant delay can be obtained even when the solutions do not follow a Gray code order. Hence, it is strictly more general than the previous definition.

Constant amortized time   An algorithm is in constant amortized time (CAT) if its total time divided by the number of solutions is constant, i.e. its average delay is constant.

Example 8. Consider the generation of the integers between 0 and $2^n - 1$ in the usual order, represented as an array of bits. Adding one to an integer 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, at most two operations per solution is required: it is a CAT algorithm. If we generate the integers in another order like the Gray code order, it possible to obtain a constant delay algorithm [Knuth 2011].

See Ruskey's book [Ruskey 2003] for many other examples of CAT algorithms. Uno's push-out method helps turn backtracking algorithms into CAT algorithms, see [Uno 2015] for the method and several of its applications.

Several problems admitting a CAT algorithm, such as the enumeration of unrooted trees [Wright *et al.* 1986], have later been proved to admit a proper constant delay algorithm [Nakano & Uno 2003]. Amortization may be used to turn a CAT algorithm into a 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. Then, by 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. This method is sometimes called Feder's trick, as he introduced it to generate the models of a 2CNF [Feder 1994].

Open problem 5.4. *Give a sufficient condition to turn CAT algorithms into constant delay algorithms, possibly using exponential space. The method of Theorem 5.1 transforms a flashlight search in constant average delay into a (strong) linear delay algorithm: under which condition can it be made constant?*

Constant amortized time is stricly more general than constant delay, as shown in the next proposition.

Proposition 5.2. *There is* ENUM·A *which can be solved in constant amortized time and polynomial preprocessing but not in constant delay and polynomial preprocessing.*

Proof. Let L be a problem, which can be solved in $O(2^n)$ but not in $o(2^n/n)$. We define $A(x)$ as $\{0i \mid i \leq |x|\}$ if $x \in L$ and $\{1i \mid i \leq |x|\}$ otherwise. The problem can be solved in constant amortized time, by computing whether $x \in L$ in time $O(2^{|x|})$ and then generating all solutions using Gray code in total time $O(2^{|x|})$. If ENUM·A can be solved in constant delay and polynomial time preprocessing, then the first solution is generated in polynomial time, which implies that L is solved in polynomial time, a contradiction. $\square$

Fixed parameter constant delay   When doing query evaluation, the query is often considered to be of fixed size, we say that we consider the *data complexity*. Hence, the number of free variables of the query is fixed, let us denote it by k. A solution is a k-tuple of elements from the structure on which the query is evaluated. Let n be the size of the structure, which is the instance of the problem, 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, it is not enough to ask the preprocessing to be in polynomial time. The choice is often to consider a linear (or almost linear) preprocessing. Since the solutions are of a fixed constant size, we could use for these problems the notion of *strong polynomial delay*, which is more demanding than constant delay: the delay should be polynomial in k, while it can be any function in k for (fixed parameter) constant delay.

Durand and Grandjean have shown that FO queries over bounded degree structures can be enumerated with constant delay and linear preprocessing [Durand & Grandjean 2007]. Following that result, many query languages have been proven to have a constant delay algorithm, see the survey of Segoufin [Segoufin 2015] and Durand [Durand 2020].

The class of enumeration problems solvable with constant delay is stable under cartesian product. However, to be stable under union, either the union should be disjoint, or the CHECK·A problem must be in constant time. These observations can be generalized by considering circuits of disjoint unions and cartesian products, representing a set of solutions, which can be enumerated with linear preprocessing and constant delay when the solutions are sparse. This approach has been investigated in [Amarilli *et al.* 2017], and can be applied to show that assignments of MSO formulas, with free variables of the first order only, can be enumerated with constant delay over structures of constant tree-width.

5.3   Random Sampling

As a way to escape the time and space required to generate all solutions, we could only generate one. We may then ask for a solution at some position in a given order, an approach considered in Chapter 6 on ordered enumeration problems. An approach, which does not require to have an a priori on the generated solutions, is to generate a representative solution, which can be achieved by uniformly sampling the set of solutions. A **polynomial time uniform generator** for the problem ENUM·A generates an element of $A(x)$, with uniform probability, in time polynomial in $|x|$. The existence of such generator is deeply connected to the possibility of approximately counting the set of solutions, see [Jerrum 2003].

The space of solutions of an enumeration problem can often be represented by some connected supergraph. As explained in Chapter 3, a traversal of the graph produces all solutions but often requires exponential space. If there is a random walk on the graph of solutions, whose stationary distribution (probability of presence on each vertex in the limit) is almost uniform, then it can be used as a random generator. When the random walk mixes in polynomial time, i.e. approximates its stationary distribution up to a constant in polynomial time, then we obtain a polynomial time random generator, which avoids the exponential space use of the supergraph method.

When the space of solutions to enumerate is infinite and even continuous, sampling is often the only approach. In [Peyronnet *et al.* 2012], we try to decide when two non deterministic automata or two Markov decision processes are far enough for some distance. When they are different, we want to explain to the user why these automata or MDPs are different, hence we want to enumerate words or strategies which prove their separation. For complexity reason, we use an approximate representation by the polytope of the set of accepted words or the polytope of the stationary distributions of the strategies. There are an *infinite number* of points in the symmetric difference of two polytopes, hence instead of enumerating them, we sample them uniformly, using a polynomial time random walk on polytopes [Lovász & Simonovits 1993].

A polynomial time uniform generator can be turned into an exhaustive enumeration algorithm solving the enumeration problem with *high probability*. The algorithm is simple: draw repeatedly solutions using the uniform generator. The only difficulty is to know when to stop drawing solutions because there is no new solution anymore. The time between the last new solution generated and the end of the algorithm can be bounded using the Coupon Collector Theorem [Erdos & Rényi 1961], so that the algorithm works with high probability. This methods has been first presented in [Goldberg 1991] and improved in [Capelli & Strozecki 2019].

Theorem 5.2. *If* ENUM·$A \in$ EnumP *has a polytime uniform generator, then* ENUM·A *can be solved with high probability by an algorithm in polynomial delay.*

The algorithm used in Theorem 5.2 needs an exponential space for storing the solutions and for the amortization of the incremental linear time. In general, we cannot get rid of the space, since Goldberg has given a tight lower bound on the product of space and delay [Goldberg 1991]. However, if *repetitions of solutions* are allowed, we can build an exhaustive algorithm using polynomial space and incremental linear time [Capelli & Strozecki 2019].

To do that, we must decide when to stop drawing solutions, without being able to maintain explicitly the set of found solutions. There exist data structures which approximate the cardinality of a dynamic set using only a logarithmic number of bits in the size of the set [Flajolet & Martin 1985, Kane *et al.* 2010]. The idea is to apply a hash function to each element seen and to remember an aggregated information on the bits of the hashed elements. Such data structures can be used to design a randomized incremental linear algorithm from a uniform generator.

Theorem 5.3. *If* ENUM·A ∈ EnumP *has a polytime uniform generator, then* ENUM·A *can be solved with high probability by an algorithm in incremental linear time* **with repetitions** *and* **polynomial space***.*

Note that Theorem 4.1 does not allow to amortize incremental linear time into a polynomial delay in Theorem 5.3. Indeed, the algorithm is randomized and the different copies of the enumeration algorithms used in the amortization would produce different solutions!

Open problem 5.5. *There are several methods to obtain a uniform generator: random walk on a reconfiguration graph [Nishimura 2018] or Boltzman samplers [Duchon et al. 2004]. Can we get an efficient enumeration algorithm directly from these methods without building the random generator?*

Ordered Enumeration

Contents

In this chapter we briefly explore ordered enumeration, that is producing a set of solutions in some order.

6.1 Specified Order

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. To make this approach more interesting, we can fix some relevant 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 [Eppstein *et al.* 2015]. In this survey, methods for generating k-best solutions are presented, which are variations on methods also used for enumeration: binary partition or the $I \cup \{j\}$ method [Lawler *et al.* 1980].

Fixed Order Enumeration problems can be further constrained by requiring the solutions to be produced in a given order. The order can either be given in the input as a black box or with a succinct definition, e.g. when the solutions are sequences of elements, giving an order on the elements yields a lexicographic order on solutions. Alternatively, the order (or family of orders) can be fixed by the problem, which is the point of view adopted here. We also require that the order can be decided in polynomial time. Since partial orders are allowed, it is a strict generalization of enumeration problems as defined in this thesis. The complexity measures and complexity classes are defined for ordered enumeration problems as they have been defined for regular enumeration problems. From a practical point of view, ordered enumeration 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, presented in the previous paragraph.

From a structural complexity perspective, adding an order constraint has no effect on the class OutputP since the solutions can be sorted after having been output in time polynomial in the solution set, as remarked in [Johnson *et al.* 1988]. However, this method stores all

solutions before sorting them, which may require exponentially more space than the algorithm which does not respect order. This space can be traded for time, by doing a selection sort. The algorithm solving the unordered enumeration problem is run to produce all solutions and we maintain s the smallest produced solutions during its run. At the end of the run, s is output, and the algorithm is run again to find the smallest solution larger than s and so on. This method multiplies the total time by the number of solutions, hence if the original algorithm is in OutputP, the algorithm for the ordered problem is also in OutputP without using more space.

It is easy to enumerate solutions in lexicographic order, when there is a flashlight search for the problem, by doing the traversal of the tree of partial solutions following the order. However, when there is no flashlight search, as for the generation of maximal independent set, a lexicographic order may dramatically change the complexity. Maximal independent sets can be enumerated in lexicographic order with polynomial delay but not in reverse lexicographic order [Johnson *et al.* 1988] if P $\neq$ NP! Likewise, maximal acyclic subhypergraphs of a hypergraph can be enumerated in polynomial delay [Daigo & Hirata 2009] but finding the first lexicographic maximal acyclic subhypergraph is coNP-hard, therefore this enumeration problem in lexicographic order is not in IncP. It turns out that many ordered enumeration problems are not in IncP because finding the first solution is NP-hard. In particular when the order is the solution size, the enumeration of many kinds of subgraphs is hard: independent sets, dominating sets, cycles . . . The Hamming weight of a solution also induces an interesting order in the context of assignements of formulas, see [Creignou *et al.* 2011, Schmidt 2017]

Hardness results with a fixed order are often directly derived from the hardness of finding some largest solution and are not specific to enumeration. Indeed, even trivial enumeration problems become hard when paired with the appropriate order (see Section 2.4 of [Strozecki 2010b]). We design a total order $<$, such that $a < b$ can be decided in polynomial time, but the enumeration of 2^n trivial solutions of an instance of size n is not in incremental polynomial time in this order, while the unordered problem can be solved in constant delay. Hence, separations of classes of ordered enumeration problems are simpler and cleaner, as stated in the following proposition.

Proposition 6.1 ([Strozecki 2010b]). *For ordered enumeration problems, P $\neq$ NP is equivalent to any of these separations*

$$\text{DelayP} \subsetneq \text{IncP} \subsetneq \text{OutputP} \subsetneq \text{EnumP}.$$

6.2 Ranked Solution

To avoid to deal with the time and space needed to generate the whole solution set, the best is to generate solutions on demand. Let A be a binary predicate, let $<$ be some total order on $A(x)$, then RANKEDSOL·$(A, <)$ is the problem of computing y from the instance (x, j), such that y is the jth element of $A(x)$ according to $<$ or a special symbol if $|A(x)| \leq j$. Problem RANKEDSOL·$(A, <)$ is also called the jth solution problem [Bagan 2009]. From that problem, we define the complexity class QueryP (see [Strozecki 2010b]), which is the set of enumeration problems ENUM·A, for which there is an order $<$ such that RANKEDSOL·$(A, <) \in$ P. Note

that QueryP is a restriction of the class NextP: instead of accessing the solutions as in a list by a next pointer, we access them directly as if they were stored in an array.

Example 9. We define LINEARSYSTEM$((A, b), x)$ the predicate which is true if and only if $Ax = b$, with A a $n * m$ matrix over some finite field $\mathbb{F}$ and x and b vectors. We show that ENUM·LINEARSYSTEM $\in$ QueryP. To solve ENUM·LINEARSYSTEM, one should output all solutions of a linear system, that is a vector space. A basis $(y_1, \ldots, y_k)$ of the vector space can be computed in polynomial time, for instance using Gaussian elimination. The set of solutions is the set of vectors equal to $\lambda_1 y_1 + \cdots + \lambda_k y_k$ for all $(\lambda_1, \ldots, \lambda_k) \in \mathbb{F}^k$. We fix an arbitrary order on $\mathbb{F}$, and consider the associated lexicographic order $<$ on the solutions seen as sequences $(\lambda_1, \ldots, \lambda_k)$. The problem RANKEDSOL·(LINEARSYSTEM, $<$) can be solved in polynomial time since the jth element of $<$ is easy to compute in polynomial time and from that, the solution is obtained by a linear number of scalar multiplications and vector additions.

If RANKEDSOL·$(A, <) \in \mathsf{P}$, the problem of counting the solutions is also in polynomial time. Reciprocally, if an enumeration problem is solved by a flashlight search, and if counting its solutions is in polynomial time, then the problem is in QueryP, using the lexicographic order yielded by the traversal of the tree of partial solutions. Only very simple problems are in QueryP, for instance listing the solutions of a first-order query over a bounded degree structure [Bagan *et al.* 2008], or monadic second-order query over trees or graphs of bounded branch-width [Courcelle 2009]. The order used for the case of second-order queries is not natural and is a byproduct of the algorithm. For first-order queries and lexicographic ordering, the class of problems for which RANKEDSOL·$(A, <)$ can be solved in logarithmic time is characterized using standard complexity assumptions [Carmeli *et al.* 2021].

Counting the perfect matchings of a graph is a classical ♯P-complete problem [Valiant 1979], but their enumeration can be done in polynomial delay [Uno 1997], hence if FP $\neq$ ♯P then QueryP $\neq$ DelayP. The same reasonning holds for ENUM·DNF, which is in NextP because of the flashlight search solving it in lexicographic order, but counting the solutions of a DNF is ♯P-complete.

In [Carmeli *et al.* 2020] the problem of generating solutions of conjunctive queries is studied in details under three different constraints: without order, in a random order and with random access, that is QueryP (unspecified order). By turning a uniform generator into an exhaustive enumeration algorithm by repeated sampling as in Theorem 5.2, it is easy to generate solutions in random order. However, designing a specific algorithm to generate solutions in random order seems more efficient in practice than randomly sampling solutions [Carmeli *et al.* 2020]. Since any problem in QueryP has a random uniform generator in polynomial time, generating in random order is easier than random access. The sets of free-connex acyclic conjunctive queries, which can be enumerated/sampled in polylogarithmic delay/time after linear preprocessing form a strict hierarchy, showing that these notions are different even for a very restricted problem.

Restricted Formalisms and Classification

Contents

We want to precisely characterize the complexity of enumeration problems. We report the very few known lower bounds and hardness results. We then study restricted formalisms of several kinds to be able to prove stronger results. We either restrict the family of enumeration problems we consider, or we weaken the computation model, or we even fix the algorithmic method used to solve an enumeration problem. This allows to prove lower bounds, to classify problems according to their complexity or to give simpler characterizations of their complexity. We also present several interesting algorithms, often derived from flashlight search, to capture some tractable cases in the several presented formalisms.

7.1 Hardness and Lower Bound Results

All problems derived from NP-complete problems are in EnumP but not in OutputP because deciding whether there is a solution is hard. In this section, we are interested in problems hard for lower complexity classes like IncP, for which deciding whether there is a solution is easy.

Hard problems   Recall that, by Proposition 3.2, a problem $\textsc{Enum}{\cdot}A \in \mathsf{IncP}$ if and only if $\mathsf{AnotherSol_A} \in \mathsf{FP}$. Hence, if we assume $\mathsf{P} \neq \mathsf{NP}$ and that $\mathsf{AnotherSol_A}$ is NP-hard, then $\textsc{Enum}{\cdot}A \notin \mathsf{IncP}$. This has been used to prove that several problems are IncP-hard: listing maximal models of a Horn formula [Kavvadias *et al.* 2000], listing vertices of a polyhedron [Khachiyan *et al.* 2006] and dualization in lattices given by implicational bases [Defrain & Nourine 2019, Defrain *et al.* 2021].

Class from a problem   To prove hardness results, we may assume complexity hypotheses on decision problems as in the previous paragraph. An alternative is to base 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 a hypergraph, a problem not known to be in OutputP after 40 years of research. Among these problems the dualization of a monotone boolean formula, the generation of vertices of an integral polytope [Boros *et al.* 2009], the enumeration of minimal keys, the enumeration of minimal dominating sets [Kanté *et al.* 2014], see [Eiter & Gottlob 1995, Hagen 2008] for many examples. In fact, a recent article [Defrain & Nourine 2019] classifies dualization problems as either in IncP or as hard as the enumeration of minimal transversals.

  We can also use the conjectures made on the complexity of Enum·DNF in Chapter 5 as a source of hardness. The class SDelayP is stable under the reduction given in Definition 8. Hence, if Enum·DNF can be reduced to some problem Enum·A, it is a clue that Enum·A might not be in SDelayP. We have done such kind of reduction from Enum·DNF over monotone formulas in [Mary & Strozecki 2019], but it does not give much information, because the problem has now been proved to be in SDelayP^{poly} [Capelli & Strozecki 2021a, Capelli & Strozecki 2021b].

Open problem 7.1. *Find a good problem on which to base hardness for* DelayP*. It should be in* IncP$_k$*, with* $k > 1$*. Plausible candidates are enumeration of minimal hitting sets in 3-uniform hypergraphs or enumeration of the circuits of a binary matroid.*

Open problem 7.2. *Can we transfer hardness between classes? Prove that some problem, for instance* Enum·DNF*, is not in* SDelayP *if and only if some other problem, for instance generating minimal transversals, is not in* IncP*.*

Strong complexity hypothesis   Instead of assuming P $\neq$ NP, we may need to make stronger assumptions. If we assume that 3SUM, the problem of deciding whether three numbers in a list sum to zero, cannot be solved faster than $O(n^2)$, there are several lower bounds on the complexity of generating all triangles of a graph [Kopelowitz *et al.* 2016]. 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 conjunctive query: either it is free-connex and its assignments can be enumerated in constant delay and linear preprocessing or it is not and its assignments cannot be enumerated with constant delay and linear preprocessing, unless the product of boolean matrices can be computed in $O(n^2)$ [Bagan *et al.* 2007]. The Strong Exponential Time Hypothesis (SETH) is a typical hypothesis in fine graind complexity, wich states that there is no algorithm solving SAT in time $2^{\alpha n}$ with $\alpha < 1$. 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 [Conte *et al.* 2019], where t denotes the number of solutions and q denotes their maximal size.

Open problem 7.3. *Can we prove the same lower bound for maximal solutions of strongly accessible set systems, but with* ETH *as an hypothesis, using the amplification method of Theorem 3.1?*

Open problem 7.4. *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.*

7.2 Hardness for Specific Algorithms or Computation model

Extension problem   The most common method to obtain a polynomial delay algorithm is to prove that EXTSOL·A is polynomial, which implies that flashlight search has polynomial delay, as explained in Chapter 3. Hence, showing that EXTSOL·A is NP-complete rules out the use of backtrack search. Extension problems have been studied for their own sake and there are many NP-hardness results, see [Casel *et al.* 2019b, Casel *et al.* 2019a] and the references therein. In particular, the extension problem of minimal transversals is NP-hard [Boros *et al.* 1998].

The only weakness of this approach, is that for some problems, the input of the extension problem solved when going through the tree of partial solutions may not be arbitrary, since we can decide in which order the extension is done. Typically, if a solution is a set of elements, the order in which elements are added (or forbidden) depends on the designer of the flashlight search and can even be different in every branch of the tree. Most of the time, this degree of liberty is not exploited but there are a few examples where it is relevant. For instance, to enumerate models of a matched CNF formula (see [Savický & Kučera 2016] for a definition), a specific variable must be fixed at each step of the flashlight search, so that the subformula is still a matched formula [Savický & Kučera 2016].

Maximal cliques   Even when we have an algorithm with good theoretical guarantees, it can be interesting to lower bound the complexity of the algorithms used in practice to solve it. For instance, linear programs can be solved in polynomial time [Karmarkar 1984] but the simplex mehod which is often used in practice has been proven to be at least in subexponential time for many pivot choices [Friedmann *et al.* 2011].

The problem of generating maximal cliques is an important problem, used to compute subgraph isomorphism. While there are many polynomial delay algorithms (see the references in [Conte *et al.* 2016]), algorithms with no guarantee on their delay such as Bron and Kerbosch algorithm [Bron & Kerbosch 1973] and CLIQUES [Tomita *et al.* 2006] are often used to find maximal cliques and are considered efficient in practice. In a recent paper [Conte & Tomita 2021], Conte and Tomita revisit these two algorithms to prove that they do not have polynomial delay. The Bron and Kerbosch algorithm (and one of its variants) have a delay at least $O(3^{n/3})$, using the same reasonning as for the total time. For the CLIQUE algorithm, they are able to prove that if it has polynomial delay then P = NP, by using it to solve the extension problem for maximal clique, which is NP-hard. It is similar to the notion of NP-mighty algorithm [Disser & Skutella 2018], i.e. some algorithms such as the simplex, can solve an NP-hard problem inside their computation even if they are used to solve a polynomial time problem.

Equivalence to a simpler enumeration problem   A method to generate maximal elements of independent set systems (closed by inclusion), in particular the maximal cliques of a graph, has been proposed by Lawler et al. [Lawler *et al.* 1980]. The method relies on solving the $I \cup \{j\}$ problem: given I an independent set maximal among the independent sets included in $\{1, \ldots, j-1\}$, compute all independent sets maximal within $I \cup \{j\}$. If this problem can be solved in output polynomial time, then an output polynomial algorithm exists for the general problem. Hence, the complexity of this method can be characterized through the complexity of a simpler (but similar) problem, which could help to prove that it cannot be applied. Note that this method yields a polynomial delay algorithm to generate maximal cliques, while the extension problem is NP-hard [Lawler *et al.* 1980], meaning that flashlight search cannot solve it in polynomial delay.

The enumeration of maximal induced subgraphs satisfying some hereditary property $\mathcal{P}$ is a generalization of enumeration of maximal sets in independent set systems. In [Cohen *et al.* 2008], the *input restricted problem* is introduced: list 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 and only if the input restricted version is in polynomial delay (similar criteria can be given for IncP and OutputP). This has been further generalized to t-restricted problems, see [Cao 2020]. While these characterizations could help prove lower bounds, they seem more suited to help design efficient enumeration algorithms (many of which are given in [Cao 2020]).

Strongly accessible set systems further generalize the previous frameworks, and their maximal elements can be enumerated using polynomial space, see [Conte *et al.* 2019] and its introduction for a nice description of the relationship between most algorithmic methods mentioned in this section.

Uniform generator   To turn a uniform generator into an enumeration algorithm using Theorem 5.2, we need to store all enumerated solutions, hence space may be exponential. Indeed, it seems necessary to encode the set of already generated solutions and these sets are in doubly exponential number and thus cannot be encoded in polynomial space. Therefore the enumeration algorithm needs time to rule out a large number of possible sets of generated solutions.

This idea has been made precise by Goldberg (Theorem 3, p.33 [Goldberg 1991]): the product of the delay and the space of an enumeration algorithm obtained from a uniform generator is lower bounded by the number of solutions to output up to a polynomial factor. On the other hand it is easy to build an enumeration algorithm with such space and delay, by generating solutions by blocks in lexicographic order (Theorem 5, p.42 [Goldberg 1991]). The proof of the lower bound relies on the fact that the enumeration algorithm can only output solutions which are given by calls to the generator: we are in a restricted black box model. A set of possible initial sequences of output elements in the enumeration is built so that its cardinality is bounded by an exponential in the space used and that the enumeration produces one of these sequences with high probability. Then, if the delay is too small, with high probability the calls to the generator have not produced any of those special sequences which proves the theorem.

However, if we allow *unbounded repetitions* of solutions, thus relaxing the constraint on the enumeration algorithm, we can devise an incremental linear algorithm with polynomial space, see Theorem 5.3.

Jumping automaton on graphs   Several enumeration algorithms for solving FO queries over sparse models (see [Vigny 2018]) can be executed by a jumping automaton on graphs or JAG [Cook & Rackoff 1980]. In this model, the input, usually a graph $G = (V, E)$, is represented by its incidence graph I, whose vertices are $V \cup E$ and with an edge (v, e) if the vertex v is incident to the edge e in G. The graph I is equipped with a unary function, which maps a vertex in V to the next vertex in V to represent the list of all vertices. For each vertex, we have access to the list of its incident edges by a function giving the first edge and one giving the next one. Finally, a constant number of unary relations and functions can be computed from G and added to the structure I. A unary relation should be interpreted as a color of elements in $V \cup E$ and a unary function as a colored edge on the vertex set $V \cup E$.

The JAG is an automaton, it has a fixed number of states of pebbles and a list of transitions. A transition of a JAG depends on the elements on which its pebbles are placed and its state. In particular, it can test whether two pebbles are on the same element or if they are of some given color. The transition moves the pebbles along edges of the structure I. A JAG solving an enumeration problem has output states and a given subset of its pebbles represents the output solution when an output state is reached in the computation. A JAG with a fixed number of pebbles models an almost constant memory. To define constant delay enumeration, we require that the delay between two outputs of a JAG is constant. We hope to prove lower bounds for constant delay enumeration using JAGs, in fact they have been introduced to prove space lower bounds for problems like connectivity [Wigderson 1992].

Proposition 7.1. *There is no JAG, which receives as input a structure I representing a graph G and u, v two vertices of G, and decides whether (u, v) is an edge in G in constant time.*

Proof. Assume that there is a JAG J, such that from any graph G a structure I can be built, on which J decides in constant time whether two elements are an edge in G. It means that the computation begins with two pebbles on u and v and J accepts after a constant number of transitions if and only if $(u, v) \in E$. Assume that all elements of I are labeled by a distinct label of size $O(\log(n))$, where $n = |V \cup E|$. In constant time, a JAG explores only a constant number of elements around u and v, hence its decision to accept (u, v) as an edge depends on an information which can be encoded into $C \log(n)$ bits, for some constant C. The labeling of the vertices is necessary to differentiate vertices, since a JAG can test whether two pebbles are on the same vertex. We have seen that the set of edges of G can be represented by the neighborhood in I of each vertex $v \in V$, hence it can be encoded by $Cn \log(n)$ bits. As a consequence, there are at most $2^{Cn \log(n)}$ different graphs represented by J and any incidence structure I. Since there are at least $2^{n^2}/n!$ unlabeled graphs, which is asymptotically larger than $2^{Cn \log(n)}$, then there is no J, which decides the edge relation in constant time.   □

We want to prove a lower bound for an enumeration problem, using an approach similar to the one used for the edge relation. However, the enumeration of edges by a JAG is trivial,

by enriching I with a function representing the list of edges. Hence, we need to consider a more complex FO query, such as one defining paths of length 2.

Open problem 7.5 (Suggested by Segoufin, personal communication). *Unconditionally prove that there is some simple FO query, say $\exists z(E(x,z) \land E(z,y))$, which cannot be solved using a JAG with a constant number of pebbles and constant delay. The preprocessing to compute the structure I representing the graph may be further restricted.*

Unconditional lower bounds are extremely hard to obtain: after fifty years of research, it is not even known whether SAT requires superlinear time to be solved. To obtain unconditional lower bounds in enumeration, the easiest approach is to show that a problem is not in some small complexity class, like constant delay with linear preprocessing, and even to restrict the computational model. That is why we have proposed Open Problem 7.5 on the JAG computational model. Indeed, unconditional separations from classical complexity involve classes of problems solvable by a family of boolean circuits: strict hierarchy inside AC^0 [Hastad 1986], polynomial size monotone circuits cannot decide the existence of a k-clique [Razborov 1985] and $\mathsf{ACC} \neq \mathsf{NEXP}$ [Williams 2014].

Open problem 7.6. *Can we define meaningful circuit classes for enumeration and prove unconditional separations of* DelayP *from one of these classes? Can we relate circuit classes for enumeration to parallel computation or descriptive complexity?*

7.3 Restricted Classes of Problems

We now present several families of enumeration problems, which are easier to classify but nonetheless contain many interesting problems. Saturation under set operators, polynomial interpolation and queries with second order variables are presented in more details because I have worked on them.

7.3.1 Constraint Satisfaction Problems

Constraint satisfaction problems are a broad generalization of the 3-SAT problem: an instance of a constraint satisfaction problem (CSP) is given as a conjunction of constraints over variables. The constraints are relations over a domain, usually finite and come from a set of constraints called the constraint language, e.g. the disjunctions over three literals for 3-SAT. A solution of a CSP is an assignment of the variables which satisfies all constraints. A CSP can also be interpreted as finding a homomorphism between two structures representing respectively the variables and the domain.

The fundamental question is to classify the constraint languages for which the problem of finding a solution is easy or hard. It turns out that in the boolean case, all constraint languages either yield a CSP in P or in NP [Schaefer 1978] and this result has been recently extended to any finite domain [Bulatov 2017, Zhuk 2020], which is surprising since in general, there are problems of intermediate complexity between P and NP by Ladner's theorem [Ladner 1975].

A CSP problem can easily be turned into an *enumeration CSP* by asking to generate all of its satisfying assignments rather than deciding whether there is one. Over a boolean domain,

there is a dichotomy: for Horn relations, anti-Horn relations, affine relations and bijunctive relations, the enumeration is in DelayP, while all other problems are IncP-hard (and even EnumP-complete) [Creignou & Hébrard 1997]. This result has been generalized to domains of larger size [Schnoor & Schnoor 2007], with a new efficient algorithm and some hardness results, but no complete classification. In particular, it is shown that some enumeration CSPs in DelayP cannot be solved by a flashlight search, contrarily to the boolean case.

Open problem 7.7. *Is it possible to prove a dichotomy for enumeration CSPs and finite domain, using results of [Bulatov 2017, Zhuk 2020]?*

The enumeration CSPs have also been considered with an order constraint on the output of solutions, as in Chapter 6. When the solutions must be given in increasing or decreasing Hamming weight order, there is also a dichotomy proved in [Creignou *et al.* 2011, Schmidt 2017].

Variations on enumeration CSP, where the constraint language is unrestricted but the instances are restricted, are studied in [Bulatov *et al.* 2012]. Typically, the problems are in DelayP when the input has a small tree-width and it seems to be different from the same results for decision problems, see [Greco & Scarcello 2013] for an overview of results on variants of enumeration CSPs.

7.3.2 Saturation by Set Operators

As explained in Chapter 3, all algorithms in incremental polynomial time can be interpreted as saturation algorithms. In particular, saturation problems, which are defined as computing the closure of a set of elements by a k-ary operation computable in polynomial time, are in IncP_{k+1}. A natural question is to understand for which operation, the defined saturation problem is in fact in $\text{IncP}_1 = \text{DelayP}$. We tried to settle this question with Arnaud Mary in [Mary & Strozecki 2016a, Mary & Strozecki 2016b, Mary & Strozecki 2019], by restricting the set of saturation operators.

In the proposed framework, a solution is a vector over some finite set, the saturation operations are of fixed arity and they act *component-wise* and in the same way on each component of the vector. We fix a finite domain D. Given a t-ary operation f (a function from D^t to D), f can be naturally extended to a t-ary operation over vectors of the same size. Let $(v^1, \ldots v^t)$ be a t-uple of vectors of size n, f acts component-wise on it, that is for all $i \leq n$, $f(v^1, \ldots, v^t)_i = f(v_i^1, \ldots, v_i^t)$. For instance, if f is the conjunction on the boolean domain ($D = \{0, 1\}$), then f acts on two vectors, which can be interpreted as subsets of $[n]$, and computes their intersection.

Definition 10. *Let $\mathcal{F}$ be a finite set of operations over D. Let $\mathcal{S}$ be a set of vectors over D. Let $\mathcal{F}^i(\mathcal{S}) = \{f(v_1, \ldots, v_t) \mid v_1, \ldots, v_t \in \mathcal{F}^{i-1}(S) \text{ and } f \in \mathcal{F}\} \cup \mathcal{F}^{i-1}(\mathcal{S}) \text{ and } \mathcal{F}^0(\mathcal{S}) = \mathcal{S}$. The closure of $\mathcal{S}$ by $\mathcal{F}$ is $Cl_{\mathcal{F}}(\mathcal{S}) = \cup_i \mathcal{F}^i(\mathcal{S})$.*

Notice that $Cl_{\mathcal{F}}(\mathcal{S})$ is the smallest set which contains $\mathcal{S}$ and which is closed by the operations of $\mathcal{F}$. As an illustration, assume that $D = \{0, 1\}$ and that $\mathcal{F} = \{\vee\}$. Then the elements of $\mathcal{S}$ can be seen as subsets of $[n]$ (each vector of size n is the characteristic vector of a subset of $[n]$) and $Cl_{\{\vee\}}(\mathcal{S})$ is the closure by union of all sets in S. Let $\mathcal{S} =$

$\{\{1, 2, 4\}, \{2, 3\}, \{1, 3\}\}$ then

$$Cl_{\{\vee\}}(\mathcal{S}) = \{\{1, 2, 4\}, \{1, 2, 3, 4\}, \{2, 3\}, \{1, 3\}, \{1, 2, 3\}\}.$$

For each set of operations $\mathcal{F}$ over D, we define the problem ENUMCLOSURE$_\mathcal{F}$: given a set of vectors $\mathcal{S}$, list the elements of $Cl_\mathcal{F}(\mathcal{S})$. Note that ENUMCLOSURE$_{\{\vee\}}$ is the problem ENUM·UNION introduced in Chapter 3. A naive saturation algorithm solves ENUMCLOSURE$_\mathcal{F}$: the elements of $\mathcal{F}$ are applied to all tuples of $\mathcal{S}$ and any new element is added to $\mathcal{S}$, the algorithm stops when no new element can be produced. If the largest arity of an operation in $\mathcal{F}$ is k then this algorithm adds a new element to a set of t elements (or stops) in time $O(t^k P(n))$ where n is the size of a solution and $P(n)$ the time to evaluate an operation of $\mathcal{F}$. Hence, ENUMCLOSURE$_\mathcal{F} \in$ IncP$_{k+1}$.

One way of solving ENUMCLOSURE$_\mathcal{F}$ in polynomial delay is to solve the associated extension problem in polynomial time. The extension problem is equivalent to the membership problem, denoted by MEMBERSHIP$_\mathcal{F}$: given $\mathcal{S}$ a set of vectors of size n, and a vector v of size n does v belong to $Cl_\mathcal{F}(\mathcal{S})$?

The Post lattice Let $\mathcal{F}$ be a finite set of operations over D. The *functional clone* of $\mathcal{F}$, denoted by $< \mathcal{F} >$, is the set of operations obtained by any composition of the operations of $\mathcal{F}$ and projections. The sets $Cl_\mathcal{F}(\mathcal{S})$ and $Cl_{<\mathcal{F}>}(\mathcal{S})$ are the same, hence it is enough to study functional clones to understand the complexity of ENUMCLOSURE$_\mathcal{F}$. When the domain is boolean, i.e. $D = \{0, 1\}$, the functional clones form a lattice called Post's lattice (see [Post 1941, Reith & Vollmer 2003, Böhler *et al.* 2005]). We prove by reduction that, for many functional clones of this lattice, the problems ENUMCLOSURE$_\mathcal{F}$ are equivalent. We are left with 4 families of functional clones:

- The clone $E_2 = < \wedge >$ or its dual $< \vee >$. The problem MEMBERSHIP$_{E_2}$ can be solved in linear time, and we can design a flashlight search in linear delay in the input, see Chapter 3.

- The clone $L_0 = < + >$ where $+$ is the boolean addition. Again MEMBERSHIP$_{<+>}$ is in polynomial time, and the vector space it generates can be enumerated in linear delay in the size of a solution. The problem is similar to ENUM·LINEARSYSTEM presented in Chapter 6 and is in SDelayP $\cap$ QueryP. The results are similar for the clone L_2 generated by the sum modulo two of three elements.

- The clone $BF = < \vee, \neg >$, which defines a boolean algebra and $M_2 = < \wedge, \vee >$. In both cases, we can compute atoms whose combinations by $\vee$ produce all solutions, and it can be done with linear delay in a solution. Again the problem is in SDelayP $\cap$ QueryP.

- The clones $S_{10}^k = < Th_k^{k+1} >$, where Th_k^{k+1} is the threshold function over $k+1$ variables, which is equal to one if and only if at least k of its arguments are ones. Because of the Baker-Pixley Theorem [Baker & Pixley 1975], $Cl_{S_{10}^k}(\mathcal{S})$ can be characterized by all the projections of $\mathcal{S}$ over k components. Hence, MEMBERSHIP$_{S_{10}} \in$ P and the flashlight search is in strong polynomial delay. In the particular case of the majority operator $< Th_2^3 >$, a reduction to the problem of enumerating models of a $2CNF$ formula gives a linear delay algoritm, using the algorithm of [Feder 1994].

As a conclusion, we obtain the following theorem.

Theorem 7.1 ([Mary & Strozecki 2016a]). *For any finite set of boolean operators $\mathcal{F}$,* ENUMCLOSURE$_{\mathcal{F}} \in$ DelayP.

We would like to prove that the complexity of the problem is in SDelayP rather than DelayP. The only open case is the clone $E_2 = < \wedge >$ for which no strong polynomial delay algorithm is known. It is equivalent to the question ENUM·UNION $\in$ SDelayP, which is mentioned in Chapter 5.

We can also consider the complexity of the *uniform problem*, where the clone is given as input instead of being fixed. Let CLOSURETRESHOLD be the following problem: given a set $\mathcal{S}$ of vectors and an integer k decide whether the vector $\mathbf{1} \in Cl_{S_{10}^k}(\mathcal{S})$, we have proven that CLOSURETRESHOLD is coNP-complete. Hence, we cannot use flashlight search to solve the uniform problem unless P = NP.

Open problem 7.8. *Prove that the uniform problem is hard for* IncP. *The proof must rely on infinite families of clones such as* $Cl_{S_{10}^k}$.

Larger domain The first generalization of the previous result is to consider domains larger than 2. However, the lattice of functional clones gets immensly more complicated. For $|D| \geq 3$, we can generalize the polynomial cases of the boolean domain: group operations or sets of operations containing a near unanimity term (similar to the threshold function) have a polynomial time membership problem. The problem MEMBERSHIP$_{\mathcal{F}}$ is known in the universal algebra community under the name *subpower membership problem* or SMP. The method for groups also works for extensions of groups by multilinear operations such as ring or modules. For semigroups, some cases are known to be PSPACE-complete and other are in P [Bulatov *et al.* 2016, Steindl 2017].

The last case we would like to extend is the clone generated by the conjunction. A natural generalization is to fix an order on D and to study the complexity of MEMBERSHIP$_f$ with f a binary monotone operation. Let f be the function over $D = \{0, 1, 2\}$ defined by $f(x, y) = \min(x + y, 2)$. This function is monotone in each of its arguments. The problem EXACT3COVER of finding an exact cover of a set by subsets of size 3 reduces to MEMBERSHIP$_f$ [Mary & Strozecki 2016a], which proves that MEMBERSHIP$_f$ is NP-complete. However, a direct application of the supergraph method yields the following proposition.

Proposition 7.2 ([Mary & Strozecki 2016a]). *If f is an associative function of arity 2, then* ENUMCLOSURE$_f \in$ DelayP.

Open problem 7.9. *Better characterize the clones for which* ENUMCLOSURE$_f$ *can be solved by a supergraph method, in particular for arity larger than 2. Is it possible to do a reverse search to avoid exponential space?*

Non uniform operations The second natural generalization is to allow operations to act differently on each component of the vectors. First, we only allow unary operations acting on a single component by turning it to zero or to one. It roughly corresponds to listing

downward closure (or upward closure) of $Cl_\mathcal{F}(\mathcal{S})$ and for the boolean domain, we are still able to prove that MEMBERSHIP$_\mathcal{F} \in \mathsf{P}$.

If we consider unary operators acting on three componenents, then there is one operator for which the membership problem is equivalent to EXACT3COVER, which leads to the following question.

Open problem 7.10. *Is the membership problem* NP*-hard for some unary operator acting on* 2 *components?*

Maximal solution   The last generalization is to enumerate minimal or maximal elements of $Cl_\mathcal{F}(\mathcal{S})$, to reduce the number of solutions output while still producing the most interesting ones. There is no natural saturation algorithm to compute the maximal elements of $Cl_\mathcal{F}(\mathcal{S})$, contrarily to all problems studied in this section. Hence, there is no guarantee that the problem can be solved by an incremental polynomial time algorithm.

We consider the boolean domain, and the order is the inclusion of solutions. We have proved that the problems of generating maximal or minimal elements of $Cl_\mathcal{F}(\mathcal{S})$ is either trivial, or equivalent to one of the following well-known enumeration problems.

- The enumeration of the maximal models of a $2 - CNF$ formula, which can be solved in linear delay [Kavvadias *et al.* 2000].

- The enumeration of the circuits of a binary matroid (eventually with a fixed element in the circuits), which is in IncP$_2$ [Khachiyan *et al.* 2005].

- The enumeration of maximal independent sets of a hypergraph of dimension k (the size of the largest hyperedge), which is in IncP$_k$ [Eiter & Gottlob 1995].

As a conclusion, generating maximal or minimal elements of $Cl_\mathcal{F}(\mathcal{S})$ over the boolean domain is in IncP for all $\mathcal{F}$.

7.3.3   Query Problem

Let $\mathcal{L}$ be some logic, then a query $\phi \in \mathcal{L}$ is a formula of $\mathcal{L}$ with free variables. Given a model, solving this query means listing all assignments of elements of the model to the free variables which satisfy the query. The decision version of this problem is the model checking problem: decide whether a formula withtout free variables is satisfied over some model.

Small solutions   Typically, the query is of small size, while the model is huge, hence we consider the data complexity of the problem (query size is fixed), see Section 5. Since the size of a solution to a query is bounded by the number of free variables, the solutions considered are small.

The most commonly studied logic is the first order logic (FO). There are constant delay algorithms when the model on which the query is evaluated is restricted to be of bounded or low degree [Durand & Grandjean 2007, Durand *et al.* 2020], of bounded expansion [Kazana & Segoufin 2020] or nowhere dense [Schweikardt *et al.* 2018]. The last result gives in fact a dichotomy on the complexity of the enumeration of FO queries modulo a complexity hypothesis.

Theorem 7.2 ([Vigny 2018]). *Let $\mathcal{C}$ be a class closed under subgraph and assume* FPT $\neq$ AW[*], *then the problem of enumerating the models of a FO query is not in* DelayP *if and only if $\mathcal{C}$ is somewhere dense.*

See [Vigny 2018] for more details on this line of work. Note that there are lower bounds for the complexity of solving an FO query with regard to the query size (coming from the model checking problem). For instance, if FPT $\neq$ AW[*], there is no doubly exponential algorithm to solve an FO query over a bounded degree structure [Frick & Grohe 2004].

Large solutions If we consider the combined complexity of solving a query, that is the formula is part of the input and the complexity depends on its size, then we must consider simpler queries (but with larger solutions). The most studied queries are conjunctive queries (CQ): conjunction of relational atoms and equality atoms, existentially quantified. Enumeration algorithms with linear preprocessing and constant delay are known [Bagan *et al.* 2007, Bagan 2009, Brault-Baron 2013] for the class of self-join *free-connex* acyclic CQs. Modulo a strong complexity assumption, this class characterizes exactly the self-join free acyclic CQs with a constant delay algorithm. For a nice exposition of the algorithm for conjunctive queries and of the lower bound see [Berkholz *et al.* 2020]

Large solutions also appear when the query is of fixed size but the free variables are second order. Hence, we list sets of elements of the model, which are of the same size as the domain of the model. Generalizing Courcelle's theorem, solving a monadic second order query over a tree, a graph or a matroid of bounded branch width can be done in SDelayP [Bagan 2006, Courcelle 2009, Strozecki 2011]. This result cannot be generalized further, since modulo ETH and some technical conditions, the tree-width of a class of structures must be polylogarithmic to obtain an FPTmodel checking algorithm [Kreutzer 2009].

With Arnaud Durand [Durand & Strozecki 2011], we have studied first order queries with free second order variables in an attempt to find classes of tractable queries with large solutions on general models. Note that the second order variables are not monadic, they can be of any fixed arity. Since in full generality such formulas may be very expressive, we consider fragments defined by the quantifier alternation of formulas inspired by results on counting problems [Saluja *et al.* 1995].

Example 10. The formula $IS(T) = \forall x \forall y \, T(x) \wedge T(y) \Rightarrow \neg E(x, y)$ holds if and only if T is an independent set.

A hypergraph H is represented by the incidence structure $\langle D, \{V, E, R\} \rangle$ where $V(x)$ means that x is a vertex, $E(y)$ that y is an hyperedge and $R(x, y)$ that x is a vertex of the hyperedge y. The formula $HS(T) = \forall x(T(x) \Rightarrow V(x)) \wedge \forall y \exists x E(y) \Rightarrow (T(x) \wedge R(x, y))$ holds if and only if T is a hitting set of H.

For any quantifier free formula, there is an algorithm with polynomial time preprocessing and constant delay. The proof uses a reduction to a constant size CNF formula, whose models can be enumerated in constant time and are then extended using a Gray code. Note that we cannot obtain an algorithm with a polynomial preprocessing independent from the formula if FPT $\neq$ W[1], since we can express the class of k-cliques of a graph with a quantifier free formula.

For formulas with only existential quantifiers, the solutions are a non disjoint union of solutions of quantifier free formulas. Hence, we obtain a polynomial delay algorithm using the method of Theorem 3.2. Moreover, the problem is equivalent to the enumeration of models of a k-DNF formula, a problem we have recently shown to be in constant delay [Capelli & Strozecki 2021a]. Hence, the complexity of solving quantifer free queries and existential queries is the same.

There is a formula with only universal quantifiers such that the enumeration problem defined by this formula is equivalent to ENUM·SAT, which is EnumP-complete. Hence, we have obtained a complete characterization of the enumeration complexity of first order formulas with regard to their quantifier alternation. There are interesting tractable fragments of first order formulas above existential formulas. They correspond to formulas which can be reduced to the enumeration of models of a 2-CNF formula or a Horn formula. They contain the problem of generating (maximal) independent sets or dominating sets of a graph.

7.3.4   Polynomial Interpolation

In this section, we consider the interpolation of a multivariate polynomial, a classical problem from computer algebra, as an enumeration problem. The input polynomial P is given as a black box: the number of its variables is n, and we can query the black box on n values $v_1, \ldots, v_n$, which returns $P(v_1, \ldots, v_n)$. The polynomial may be defined over $\mathbb{Z}$, $\mathbb{Q}$ or a finite field. Interpolating P means computing explicitly P as a set of monomials, by using queries to compute the value of the polynomial on several points. When analyzing interpolation algorithms from the enumeration point of view, we are interested in their delay and incremental time.

While any set of objects can be mapped to the monomials of a polynomial, to obtain good enumeration algorithms we are interested in polynomials with low degree and which can be evaluated in polynomial time. Many interesting problems can be embedded in the monomials of easy to evaluate polynomials:

- the cycle-covers of a graph are the monomials of the determinant of its adjacency matrix,

- the spanning trees of a graph are the monomials of the determinant of its Kirchoff matrix by the Matrix-Tree theorem (see [Jerrum 2003])

- the spanning hypertrees of a 3-uniform hypergraph are the monomials of the Pfaffian of its Laplacian matrix, as proved in [Masbaum & Vaintrob 2002]

- the probability of acceptation of n letter words by a probabilistic automaton can be encoded into the monomials of a multilinear polynomial obtained as a product of matrices [Kiefer *et al.* 2011, Peyronnet *et al.* 2012]

To obtain a complexity depending on the number of monomials, interpolation algorithms for multivariate polynomials need to be randomized. When the complexity depends on the number of monomials, the interpolation is for *sparse polynomials* by opposition to interpolation algorithms for *dense polynomials*, with a complexity depending on the degree and the number of variables. This corresponds to the notions of input sensitive and output sensitive

algorithms in enumeration. The first algorithms designed for sparse polynomials are in output polynomial time with a small probability to fail [Ben-Or & Tiwari 1988, Zippel 1990], while we describe here more recent incremental polynomial time or polynomial delay algorithms [Klivans & Spielman 2001, Strozecki 2010a, Strozecki 2013].

The decision problem associated to interpolation is called *polynomial identity testing* (PIT): given a black box polynomial decide whether it is the zero polynomial. To solve PIT, we use a probabilistic algorithm, which has a good complexity and an error bound which can be made exponentially small in polynomial time.

Lemma 7.1 (Schwarz-Zippel [Zippel 1979]). *Let P be a non zero polynomial with n variables of total degree D, let $\varepsilon > 0$, if $x_1, \ldots, x_n$ are randomly chosen values in a set of integers S of size $\frac{D}{\varepsilon}$ then the probability that $P(x_1, \ldots, x_n) = 0$ is bounded by ε.*

The algorithms presented in the following are based on solving PIT in randomized polynomial time, transformations of the considered polynomials by substitution of variables and interpolation of univariate polynomials in polynomial time.

Polynomial Delay Algorithm   Let e be a vector of dimension n with positive integer coefficients. We denote by $\mathbf{X}^e$ the term $\prod_{i=1}^{n} X_i^{e_i}$. A multivariate polynomial is the sum of monomials, that is disctinct terms multiplied by scalar coefficients. The degree of a monomial $\lambda \mathbf{X}^e$ is $\max_{1 \leq i \leq n} e_i$, the maximum of the degrees of its variables. The total degree of a monomial is $\sum_{i=1}^{n} e_i$, the sum of the degrees of its variables. Let d (respectively D) denote the degree (respectively the total degree) of the polynomial we consider, that is the maximum of the degrees (respectively total degrees) of its monomials. We assume for the first algorithm that the polynomial is multilinear, i.e. $d = 1$ which implies that D is bounded by n.

We consider the decision problem MONOMIAL-FACTOR: an input is a term $\mathbf{X}^e$, that is a product of variables, and a polynomial P given as a black box; the input is accepted if $\mathbf{X}^e$ divides a monomial of P. This problem corresponds to the extension problem defined in Chapter 3. Problem MONOMIAL-FACTOR can be solved in randomized polynomial time for multilinear polynomials (see [Strozecki 2013]): Substitute a single variable Y to all variables in $\mathbf{X}^e$, then choose a random assignment of the other variables. The problem has a positive answer (with high probability) if and only if the degree of the obtained univariate polynomial in Y is equal to the total degree of $\mathbf{X}^e$.

From the resolution of MONOMIAL-FACTOR, we obtain a randomized polynomial delay interpolation using a flashlight search, with a few technicalities due to the randomization of the algorithm and the problem of computing the coefficient of each monomial.

Theorem 7.3 ([Strozecki 2013]). *Let P be a multilinear black box polynomial with n variables, and total degree D and let $\varepsilon > 0$. There is an algorithm which computes the set of monomials of P with probability $1 - \varepsilon$, with a delay in time $O(nD^2(n + \log(\varepsilon^{-1})))$ and $O(nD(n + \log(\varepsilon^{-1})))$ black box queries on points of size $O(\log(D))$.*

The support of a monomial is the set of variables which have degree one or more in the monomial. If the set of supports of monomials of a polynomial is such that, no support is included into another, then MON-FACTOR can be solved in polynomial time by substituting

zero to all variables not in $\mathbf{X}^e$ and solving PIT on the obtained polynomial. We can then use the same flashlight search as for multilinear polynomial.

Open problem 7.11. *Is it possible to solve* Mon-Factor *for other classes of polynomials, for instance polynomials given by read twice arithmetic formulas or depth three arithmetic circuits? Is it possible to design polynomial delay algorithms based on reverse search rather than on flashlight search?*

Polynomial Incremental Time Algorithm In the context of polynomial interpolation, the problem of finding another solution is equivalent to the problem of finding a single monomial. Indeed, given a polynomial P by a black box and an explicit subset S of its monomials, it is easy to simulate queries to $P - \sum_{M \in S} M$. Then, it is easy to grow S, until $P - \sum_{M \in S} M = 0$, which can be decided by solving PIT.

We can relax the condition of supports of monomials not included into another one presented in the previous section. Let us consider polynomials such that all the supports of the monomials are different, we say that these polynomials have *distinct supports*. In this case, a simple greedy algorithm produces a monomial in polynomial time: from the set of all variables, remove a variable while the polynomial restricted to this set of variables is non zero.

For polynomials with disjoint supports, all monomials can be easily isolated, hence finding a monomial is easy. For general polynomials, there is a clever transformation of a multivariate polynomial into a univariate polynomial, such that the monomial of lowest degree in the univariate polynomial comes from a single monomial of the multivariate polynomial (see [Klivans & Spielman 2001]). The proof relies on a generalized isolation lemma, where all variables are mapped to a different random power of a single variable.

Theorem 7.4 (Adapted from Theorem 12 of [Klivans & Spielman 2001]). *There is a randomized polynomial time algorithm which given a black box access to a non zero polynomial P with n variables and total degree D and $\varepsilon > 0$ returns a monomial of P with probability $1 - \varepsilon$ in a time polynomial in n, D and with $O(n^6 D^4)$ calls to the black box on integers of size $\log(\frac{nD}{\varepsilon})$.*

There is another method for the case of polynomials of small degree. Using a generalization of the algorithm solving Mon-Factor over multilinear polynomials, we first find a maximal power of a linear factor of a monomial. Then, we would like to do the division by this factor to recursively find a monomial in the quotient. It turns out that this quotient can be evaluated on values using substitution and univariate interpolation, which is enough for our algorithm. We obtain an algorihm to find a single monomial, which is better than the general algorithm when $d < 10$.

Theorem 7.5 ([Strozecki 2013]). *Let P be a polynomial with n variables of degree d and of total degree D and let $\varepsilon > 0$. There is an algorithm which computes a monomial of P, with probability $1 - \varepsilon$. It performs $O(nD^d)$ calls to the oracle on points of size $\log(\frac{n^2 D}{\varepsilon})$.*

Open problem 7.12. *All incremental polynomial time algorithms rely on the evaluation of all the monomials found to produce a new one. Is there a class of polynomials for which this*

set can be stored concisely and evaluated efficiently? Is there an alternative algorithm using only polynomial space?

Lower bounds   For any t evaluation points, one may build a polynomial with t monomials which vanishes at every point (see [Zippel 1990]). Therefore, if we do not have any a priori bound on t, we must evaluate the polynomial on at least $(d+1)^n$ n-tuples of integers to decide PIT with a deterministic algorithm. Hence, it is not possible to obtain a deterministic interpolation algorithm in output polynomial time for a polynomial given as a black box. Even for an explicit representation of a polynomial, such as arithmetic circuits, finding an algorithm in P for PIT would be a major achievement with many complexity implications, see e.g. [Kabanets & Impagliazzo 2004].

For multilinear polynomials, it is possible to prove that PIT is equivalent to MON-FACTOR [Strozecki 2013], so the fact which yields a randomized polynomial delay algorithm is also the obstacle to an efficient *deterministic enumeration algorithm*.

To rule out the possibility of doing a flashlight search on a class of polynomials, it is enough to prove a hardness result for MON-FACTOR. By encoding a 2-CNF formula into an arithmetic circuit, which is then homogenized, we prove that MON-FACTOR can simulate the problem of finding an assignment of a 2-CNF formula of minimal Hamming weight. This proves the NP-hardness of MON-FACTOR over a very restricted class of polynomials [Strozecki 2013]: degree two polynomials represented by a depth four arithmetic circuit.

Solving a Practical Enumeration Problem

Contents

Most concrete enumeration problems are too complex to be solved by an algorithm with a theoretical guarantee on its delay. In that case, input sensitive algorithms may be efficient in practice, more than a polynomial delay algorithm, see the example of maximal cliques [Cazals & Karande 2008]. Instead of trying to obtain the best asymptotic worst case delay and fail, it becomes worthwile to optimize for the typical size and typical input, while relaxing the constraints on exhaustivity and redundancy of solutions. In this chapter we summarize the results we obtained in [Barth *et al.* 2015]: we present an enumeration problem from cheminformatic, and an algorithm designed to solve it on small instances rather than obtaining a good complexity. We show that methods from enumeration can be combined to build an algorithm which performs well in practice, and we try to underline how the design of a practical algorithm differs from the design of an algorithm with the best possible delay.

8.1 Model

We study the problem of **generating all colored planar maps up to isomorphism** representing possible molecules obtained from a set of elementary starting motifs. A *map* is a graph equipped with, for each vertex, a cyclic order on its neighbors. We consider maps, to retain some geometric properties of the molecule we model. We want to design molecular cages which can be embbeded on a sphere and are thus planar. When the map is planar, the orders of each neighborood are given by a planar embedding of the graph. The two main difficulties of the problem are the sheer number of solutions and the generation of duplicates because we are interested by maps up to isomorphism.

The basic chemical elements are modeled by maps called *motifs* (see Figure 8.1), with a single *center* vertex which represents the element and several neighbors which represent its chemical connections, which are labeled by an element of $\mathcal{A}$. The colors are partitionned into positive and negative colors: each positive color a in $\mathcal{A}$ has a unique complementary negative
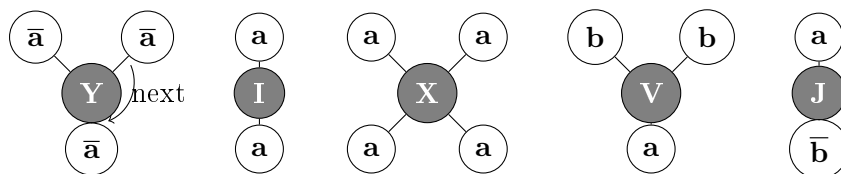
Figure 8.1: Example of motifs on $\mathcal{A}_M = \{\mathbf{Y}, \mathbf{I}, \mathbf{X}, \mathbf{V}, \mathbf{J}\}$ and $\mathcal{A} = \{a, \overline{a}, b, \overline{b}\}$.

Figure 8.2: Example of two maps of motifs based on $\mathcal{M} = \{\mathbf{Y}, \mathbf{I}\}$, the right one is saturated.

color denoted by $\overline{a}$. Each color represents a different kind of reacting center, which can only be paired with a reacting center of complementary color. Then the motifs are assembled to form a *map of motifs*, as shown in Figure 8.2, by adding an edge between two vertices from different motifs with complementary colors. A vertex which is connected only to its center vertex is said to be *free*. We are interested in *saturated* maps of motifs, that is maps with no free vertex, because they may represent real molecules. Note that, in this model, all geometric informations on the molecule represented by a motif are lost, but the order of neighbors of a vertex. Hence a map of motifs does not always represent an existing molecule.

8.2   Heuristic Algorithm

To solve such a problem, we decompose it into several simpler steps. Our algorithm works as follows:

- Generate very simple maps of motifs with efficient algorithms. Since all graphs have a spanning tree, we generate all colored ranked trees built from a given set of motifs. To do that, classical CAT algorithms to generate all rooted ranked trees can be adapted so that the produced trees respect all degrees and color constraints. This could be further improved by generating unrooted trees, by adapting CAT algorithms for this problem such as [Li & Ruskey 1999], but it is less clear how to do so.

- From a tree, with many free vertices, we need to find all possible ways to pair them so that the obtained map is planar and saturated. We call the pairing operation a *fold*. It is easy to see that the order in which vertices are paired is irrelevant. Using a dynamic program, the possibility for a fold to contain a given pairing is computed in quadratic time in the number of vertices. Then, using this information, a flashlight search gives all solutions with linear delay.

- Detect whether the generated map is isomorphic to one generated before, by comparing its signature to the signatures of the previous maps stored in a set data structure

(hash map or AVL in our implementation). If the map is new, compute several indices useful for the chemist. We consider the minimum sparsity, to represent structural soundness of the molecule, which can be computed in polynomial time on planar graphs [Park & Phillips 1993]. The equivalence classes of the vertices help to understand the symmetries and the roundness of the molecule. We are also interested in the sizes of the faces of the map, which can be used as entrance inside the molecule when used as a cage.

The problem with this three steps approach is that many useless intermediate results are generated, such as trees which cannot be folded into a molecular map. Moreover, the same map has many different spanning trees and is thus generated by folding each of these different trees obtained at step one. While the three steps have very efficient algorithms, with strong theoretical guarantees on their complexity, their combination solves our problem, but *has no bound on its delay nor its total time*.

One solution to this problem is to avoid to generate trees which cannot be folded. A necessary condition for a map of motifs to be foldable into a saturated map is to have as many free vertices of color $a \in \mathcal{A}$ as vertices of color $\bar{a}$. Such a map is called an *almost foldable map*. To a map of motifs M, we associate the statistic vector C_M, which counts the number of vertices of each type (negative colors are counted negatively). An almost foldable map has an all zero vector of statistics. It is possible to compute by dynamic programming a predicate $A(C_T, n)$, which is true if and only if there is a tree T of size n and statistic vector C_T. Using this predicate, the algorithm generating all trees can be turned into an algorithm of the same complexity generating all almost foldable trees. This makes the algorithm run faster by several orders of magnitude, since the cost of computing A, while large, is done once in the preprocessing.

The second problem is that we generate graphs with bad indices which may not be interesting for the chemist, especially because the maps generated are in too large number to be inspected manually. The trouble here is that indices of interest may change (and have changed during our work) depending on the application, the kind of instances or even our comprehension of the problem. Moreover, there is no clear cut off λ such that we can say, we want to generate all maps with a minimum sparsity larger than λ. We were not able to solve this problem, but a good approach, described in Chapter 9, is to enumerate only a subset of the solutions, which covers as best as possible the set of all solutions.

Our work leaves many theoretical questions open.

Open problem 8.1. *Can we prove that our algorithm is efficient when seen as an input sensitive algorithm, i.e. is its total time bounded by a polynomial (or even linear) in the maximal number of generated solutions.*

Is the problem in OutputP*? Representation of planar maps by trees [Aleardi et al. 2008] may be helpful to obtain a better algorithm.*

8.3 Implementing a Tool

The aim of this work was to produce a usable tool for chemists, efficient enough to generate molecular maps with ten to twenty motifs. The source code is available on https://

`yann-strozecki.github.io/textesmaths.html` and results are available on `http://kekule.prism.uvsq.fr/`. Figure 8.3 shows how these results are presented to the user. These results have been used by our colleague Olivier David, to synthesize several new molecules, as described in [Barth *et al.* 2020], see Figure 8.4 for the interpretation of a molecular map as a real molecule.

To make our tool useful, it has been quite heavily optimized and tuned. Indeed, generating all molecular maps of size 12 from three different motifs of degree three already takes 50 seconds, because it generates about 50 million trees and 10 million saturated maps, to get only 4476 distinct maps. We present three important choices made to obtain an efficient algorithm, which are common in practice but may not be natural for theoreticians.

1. **Forget exhaustiveness.** We restrict the set of enumerated solutions to decrease their number and to get more efficient enumeration algorithms. However, the enumeration is not exhaustive anymore and we may skip interesting solutions. To do that, instead of generating trees in the first phase, we generate either paths or cycles, that are folded like the trees in the second phase. It implies that we get only maps which have either an Hamiltonian path or an Hamiltonian cycle. Since small planar graphs have often such subgraphs [Holton & McKay 1988, Aldred *et al.* 1999] it is a mild restriction and it implies that the generated maps are connected or biconnected, which are relevant properties for a molecule. Generating paths instead of trees in the previous example yields 4463 distinct maps instead of 4476, but the generation time drops to less than a second.

2. **Reconsider asymptotic complexity.** The main practical bottleneck of our algorithm, taking more than 90 percent of the time, is the computation of the signature of each molecular map we obtain, to decide whether it has already been generated. There is a theoretical linear time algorithm to compute the signature of a map [Hopcroft & Wong 1974] and an almost linear time algorithm which is implementable [Hopcroft & Tarjan 1973]. However, we deal with very small maps in our work (size at most 40), hence the complex linear time algorithms are less efficient than a simpler quadratic time algorithm: from each edge of the map, do a traversal following the order of the edges, and keep the smallest traversal (sequence of vertices encountered) as a signature.

3. **Adapt the algorithm to the input.** Worst case complexity is not always relevant since the typical input may be quite different from the worst one. In our algorithm, we know that many redundant solutions are generated, hence when we compute a signature, most of the time it is already present. Therefore, we want to be able to decide quickly if a solution is redundant, but we have a lot of time budget for each new solution found. During the internship of Ruben Staub [Staub 2016], we proposed an alternative signature: the set of all traversals instead of the minimal one. When a map distinct from the previously generated is found, all traversals are computed and stored. Each time a map is generated, a *single* traversal is computed, which is enough to find whether the map has already been generated. This requires more space, but speed-ups our algorithm by a factor 10 in some cases.
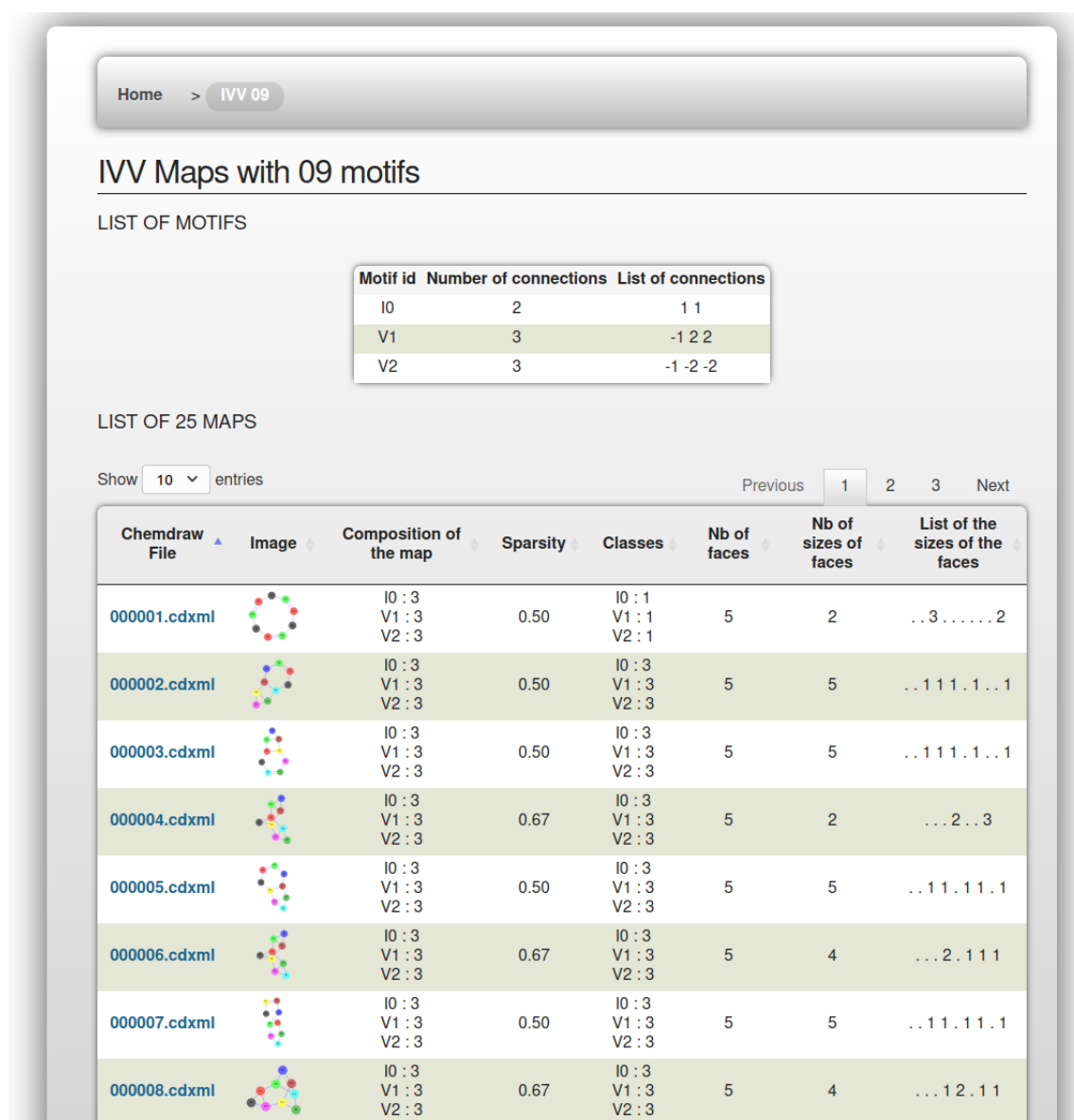
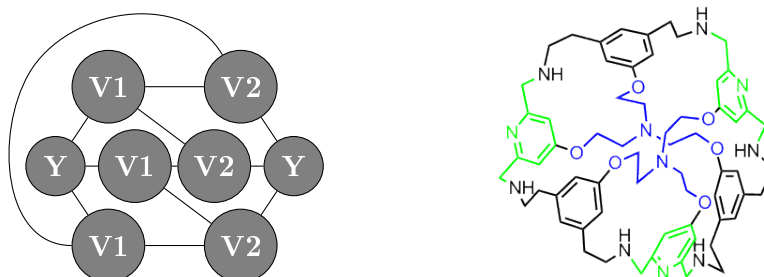Figure 8.3: Presentation of the results for a set of three motifs and a target size of 9.

Figure 8.4: A molecule derived from a map generated by our algorithm.

New Horizons for Enumeration

There are several popular areas of complexity which have been brought to enumeration and that we have not covered. For instance, it is possible to define the FPT versions of the classes presented in this thesis. Enumeration of vertex covers or weighted assignments of satisfiability problems admit enumeration algorithms with an FPT delay [Creignou & Vollmer 2015, Creignou *et al.* 2017b]. Some complexity results presented here have been adapted to FPT classes [Meier 2020]. A more restrictive way of defining kernelization for enumeration has been proposed in [Golovach *et al.* 2021]. A fully polynomial enumeration kernel characterizes problems solvable in total time bounded by a polynomial in the input times a function of the parameter, while a polynomial-delay enumeration kernel charaterizes problems solvable in FPT delay. Using these definitions, lower and upper bounds for several parametrizations of the problem of enumerating maximal (minimal) matching cuts are proved.

In dynamic problems, the input is changed incrementally and some object is maintained, for instance the maximal matching of a graph [Baswana *et al.* 2015]. This idea is a good fit for query answering, where the input is huge and the cost of preprocessing is proportional to the input but usually not the enumeration phase. Several 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 the size of each solution and update in logarithmic time [Amarilli *et al.* 2019].

Enumeration problems are especially subject to the problem of combinatorial explosion because even a constant delay algorithm cannot escape the cost of going through the whole solution space, which is often 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 amount of data. Hence, we should find ways to generate fewer solutions while still capturing the essence of the set of solutions. Following ideas of several enumeration specialists (in particular several presentations by Takeaki Uno), we present promising directions, which have been explored in the last years. The main idea is, instead of listing the whole set of solutions, to produce a *compressed* version of this set, of much smaller size but still useful.

9.1   Succinct Representation of the Solution Space

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 solution very succinctly but is not directly useful for computation.

An extreme case consists in problems which admit a polynomial time sampling algorithm or which are in QueryP. In that case, the input itself (or rather the solution generator) is a good representation of the set of solutions, since they can all be accessed easily and statistical properties of the set of solutions can easily be computed.

Another example of a large set represented in a succinct way is given by algebra: a vector space is represented by a finite basis, a group by a presentation and a matroid by its fundamental cycles. . . In all these cases, it is possible to decompress efficiently the representation, that is to enumerate the elements of these objects from the succinct representation in incremental polynomial time. In Chapter 7, we have presented the problem of computing closure under set operators (see [Mary & Strozecki 2019]), which also asks to produce from a basis all elements of a set system closed under several operations.

Sometimes, partial solutions can be trivially extended into many solutions. For instance, models of existential FO formulas [Durand & Strozecki 2011] or of k-DNF formulas [Capelli & Strozecki 2021a] are obtained by extending a partial model of constant size (a fixed number of variables are set) with all possible values of the remaining variables using Gray code enumeration. In this case, instead of exhaustively generating models, it is more efficient to generate only the partial models, as a concise representation of the set of models. This approach is relevant because the sets of solutions represented by each partial model are disjoint. A similar but more general representation of set of models is proposed and enumerated for Horn formulas [Wild 2012].

Decomposition   The two most simple operations on sets are the Cartesian product and the union and we have seen that they let enumeration classes stable in Chapter 3. Hence, it is interesting to decompose a set of solutions thanks to these two operations. It may come naturally: 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 algorithm to list spanning trees using a more involved Cartesian product decomposition is given in [Conte *et al.* 2018] as well as several examples of Cartesian decomposition for enumeration.

In algebraic complexity, Cartesian product and union correspond to product and sum: instead of representing a multivariate polynomial explicitly by a list of monomials, it can be represented by an arithmetic circuit with sum and product gates. However, this representation is hard to exploit: deciding whether the represented polynomial is zero is not believed to be in P, deciding whether a monomial has a zero coefficient is $\sharp$P-hard and counting monomials is even harder, see [Fournier *et al.* 2015]. A more tractable representation, for which most of the previous problems are simple, is to decompose a polynomial into product

of small polynomials. We have investigated with several coauthors the task of factoring the sparse representation of a polynomial [Chattopadhyay *et al.* 2021]. Interestingly, this does not always yield a smaller representation since $X^{k+1} - 1 = (X - 1) \sum_{i=0}^{k} X^i$.

The previous decompositions are reminiscent of knowledge compilation [Darwiche & Marquis 2002], where a 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. Amarilli et al. [Amarilli *et al.* 2017] 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 models of a d-DNNF. Set circuit is also essentially the same notion as factorised representation of a database [Olteanu & Závodný 2012], used to succinctly represent databases and query results. The methods of enumerating objects by first reducing the instance to compact boolean circuits is also used to obtain good practical algorithms, for instance to generate interval subgraphs [Kawahara *et al.* 2019].

Open problem 9.1. *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 9.2. *What decomposition of solution sets can be useful in enumeration outside of union and Cartesian product?*

Equivalence Relation   Another way to reduce the solution space is to define an equivalence relation, to identifiate solutions which are too similar. Then, only one representative per equivalence class is 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 [Ruskey 2003] and the problem presented in Chapter 8.

For problems whose solutions can be represented by AND/OR graphs (representing dynamic programs), there is a natural notion of equivalence for which equivalence classes can be generated with polynomial delay [Wang *et al.* 2020], with interesting applications in bioinformatics.

With Maël Guiraud and Dominique Barth, we have designed algorithms to find schedulings of periodic messages with a small latency [Barth *et al.* 2021, Guiraud 2021]. As often with optimization problems, we enumerate exhaustively all solutions to find the best one. Therefore, to make our algorithms interesting both theoretically and practically, the set of considered schedulings must be reduced as much as possible. We use two equivalence relations and a partial order to filter the set of schedulings:

- Compact representation: an equivalence class, which takes into account the combinatorial structure of a scheduling but abstracts away the precise timings.

- Canonical representation: an equivalence class, to abstract out the symmetries of a scheduling.

- Minimal solutions: the latency of all messages must be locally minimal, otherwise the scheduling is dominated and is not useful for computing an optimum.

9.2   Approximate Representation of the Solution Space

In the previous section we have explored the idea of compressing the set of solutions exactly. But we could also compress it with loss, that is producing a 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 [Papadimitriou & Yannakakis 2000]. 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 [Durand & Flajolet 2003], or with a sublinear space the minimum spanning tree or maximal matching can be approximated [McGregor 2014].

Over-approximation   Instead of approximating each solution, we may approximate the set of solutions by a larger set, outputting elements which are not far from being solutions. For the problem of enumerating minimal sets of size at most k satisfying monotone properties such as minimal vertex covers, minimal dominating sets in bounded degree graphs, minimal feedback vertex set . . . , efficient algorithms exist but they produce solutions of weight up to ck for a constant c [Kobayashi *et al.* 2020].

Query enumeration over a bounded degree structure can be done in constant delay and linear preprocessing [Durand & Grandjean 2007]. To speed-up the preprocessing, since the database cannot be read entirely anymore, a randomized algorithm must be used, as in property testing. There is a constant delay algorithm with *polylogarithmic preprocessing* instead of linear if tuples not far from a solution for the edit distance are allowed in the enumeration [Adler & Fahey 2021]. If the proportion of tuples that are answers to the query is sufficiently large, then all answers will be enumerated, by interleaving randomly sampled tuples. The idea of interleaving solutions, when there is an efficient algorithm to enumerate all solutions and that almost all potential solutions are indeed solutions has already been proposed by Leslie Ann Goldberg in a deterministic setting. She proved that all classes of graphs defined by almost sure first order property can be enumerated with polynomial delay [Goldberg 1991].

Many enumeration problems ask to enumerate minimal or maximal objects, such as maximal cliques or minimal hitting sets, to avoid less useful solutions. However, these problems are much harder and are sometimes solved by generating all non extremal objects. For instance, to generate minimal hitting set, the most efficient algorithms are based on the generation of all hitting sets, using pruning rules to avoid as most non minimal hitting set as possible [Murakami & Uno 2014].

Finally, a more restricted way to do an over-approximation is to allow repetitions of a solution instead of allowing elements which are not solutions. It is our approach when turning a random generator into an exhaustive enumeration algorithm using only polynomial space,

as explained in Chapter 4.

Under-approximation   In the previous section, the approximation is done by outputting more elements than solutions, which goes against the idea of reducing the solution space and may thus not be useful in practice. To approximate a set of solutions $A(x)$, we can use some subset S such that $|S|$ is within some factor of $|A(x)|$. I am not aware of any result of this kind, however some algorithms use the property that a large subset of solutions is easy to enumerate to have enough time to enumerate the rest. Such algorithms are used for the enumeration of clause sequences of a k-CNF formula [Bérczi *et al.* 2021] and for the enumeration of the models of a k-DNF formula [Capelli & Strozecki 2021a].

Open problem 9.3. *Design a constant factor under-approximation in incremental polynomial time for a non artificial enumeration problem not known to be in* IncP.

When designing an under-approximation, the choice of solutions in the approximation is critical. We would like them to be more interesting than the ones not enumerated. When there is a relevant order on solutions, enumerating in this order, as in Chapter 6, can be seen as a way to do meaningful under-approximation.

To be more general, we could allow both to enumerate elements which are not solutions and to miss some solutions. Let S be the set enumerated to approximate the solution set $A(x)$. To control the quality of the approximation, we say that S is an ε-approximation if $|S \triangle A(x)| \leq \varepsilon |A(x)|$. This definition is similar to the notion of approximation in knowledge compilation, for which several lower bounds are known [de Colnet & Mengel 2020].

Diversity   The approximation based on the number of solutions may not be relevant: what if we get a large family of similar solutions but miss all the diversity of the solution set?

Most of the time, there is no notion of quality over the solutions or this notion is too vague to formalize. Think about a system recommanding routes between two locations, each user may have specific desires such as only using streets she already knows or constraints too informal to be implemented such as preferring roads with a beautiful scenery. Hence, such a system should propose solutions as *diverse* as possible, to maximize the chance that one fulfills the unknown constraints of each user.

To this aim, 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 ENUM·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 have much less time than solutions. 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 then 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 is designed so that any solution is close to some solution in the cover. Hence, among the cover, there must be many solutions which are quite different, which hopefully makes such a cover useful in practice.
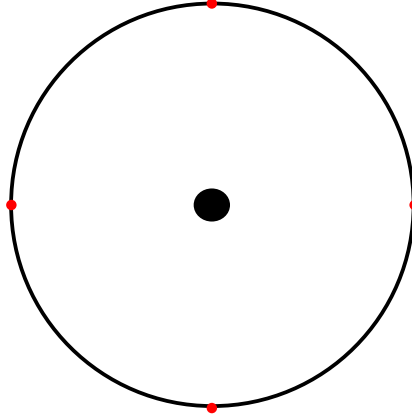
Figure 9.1: Solutions on a circle and around its center. A subset of solutions (here in red) maximising diversity always lies on the circle, missing solutions at the center.

Open problem 9.4. *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 relevant distance over solutions.*

Open problem 9.5. *Assume we have a supergraph of solutions, and the distance over solutions is related to the distance in the supergraph. What random walk on the supergraph and which property of the supergraph yield a d-approximation? Can we use the same approach with flashlight search?*

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 [Baste *et al.* 2020]. The method converts any tree-decomposition based dynamic programming algorithm into a dynamic programming algorithm for the same problem with an additional diversity constraint. However, the method is exponential in the number of produced solutions in addition to the treewidth of the graph. To find r maximally diverse spanning trees, there is a polynomial time algorithm in r [Hanaka *et al.* 2021], which allows to produce a large set of diverse solutions.

Diversity seems to well capture the notion of producing different solutions, however it does not guarantee to cover well the set of solutions, see Figure 9.1. We would like to relate large set of diverse solutions to the notion of d-approximability. To that aim, it is interesting to replace the sum of the distances between solutions by the minimum of the distances in the definition of diversity.

Open problem 9.6. *Generate r spanning trees, while maximising their minimal Hamming distances, in time polynomial in r.*

Bibliography

[Adler & Fahey 2021] Isolde Adler and Polly Fahey. *Towards Approximate Query Enumeration with Sublinear Preprocessing Time*. arXiv preprint arXiv:2101.06240, 2021. (Cited on page 72.)

[Agrawal *et al.* 1994] 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. (Cited on page 2.)

[Aho & Hopcroft 1974] Alfred V Aho and John E Hopcroft. The design and analysis of computer algorithms. Pearson Education India, 1974. (Cited on pages 8 and 10.)

[Aldred *et al.* 1999] Robert EL Aldred, Sheng Bau, Derek A Holton and Brendan D McKay. *Cycles through 23 vertices in 3-connected cubic planar graphs*. Graphs and Combinatorics, vol. 15, no. 4, pages 373–376, 1999. (Cited on page 66.)

[Aleardi *et al.* 2008] Luca Castelli Aleardi, Olivier Devillers and Gilles Schaeffer. *Succinct representations of planar maps*. Theor. Comput. Sci., vol. 408, no. 2-3, pages 174–187, 2008. (Cited on page 65.)

[Amarilli *et al.* 2017] 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. (Cited on pages 33, 39 and 71.)

[Amarilli *et al.* 2019] 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. (Cited on page 69.)

[Andrade *et al.* 2016] 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, vol. 11, no. 1, page 25, 2016. (Cited on page 3.)

[Avis & Fukuda 1996] D. Avis and K. Fukuda. *Reverse search for enumeration*. Discrete Applied Mathematics, vol. 65, no. 1, pages 21–46, 1996. (Cited on pages 4, 18, 20, 25 and 26.)

[Bagan *et al.* 2007] 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. (Cited on pages 48 and 57.)

[Bagan *et al.* 2008] Guillaume Bagan, Arnaud Durand, Etienne Grandjean and Frédéric Olive. *Computing the jth solution of a first-order query*. RAIRO-Theoretical Informatics and Applications, vol. 42, no. 1, pages 147–164, 2008. (Cited on page 45.)

[Bagan 2006] Guillaume Bagan. *MSO Queries on Tree Decomposable Structures Are Computable with Linear Delay.* In Zoltán Ésik, editor, Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings, volume 4207 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2006. (Cited on page 57.)

[Bagan 2009] Guillaume Bagan. *Algorithms and complexity of enumeration problems for the evaluation of logical queries.* PhD thesis, University of Caen Normandy, France, 2009. (Cited on pages 1, 8, 9, 10, 22, 44 and 57.)

[Baker & Pixley 1975] Kirby A Baker and Alden F Pixley. *Polynomial interpolation and the Chinese remainder theorem for algebraic systems.* Mathematische Zeitschrift, vol. 143, no. 2, pages 165–174, 1975. (Cited on page 54.)

[Barth *et al.* 2015] 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 International Publishing, 2015. (Cited on pages 3 and 63.)

[Barth *et al.* 2020] Dominique Barth, Olivier R.P. David, Franck Quessette, Yann Strozecki and Sandrine Vial. In silico discovery of complex molecular containers. 2020. (Cited on page 66.)

[Barth *et al.* 2021] Dominique Barth, Maël Guiraud and Yann Strozecki. Deterministic scheduling of periodic messages for cloud ran. 2021. (Cited on page 71.)

[Baste *et al.* 2020] Julien Baste, Michael R. Fellows, Lars Jaffke, Tomás Masarík, Mateus de Oliveira Oliveira, Geevarghese Philip and Frances A. Rosamond. *Diversity of Solutions: An Exploration Through the Lens of Fixed-Parameter Tractability Theory.* In Christian Bessiere, editor, Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020, pages 1119–1125. ijcai.org, 2020. (Cited on page 74.)

[Baswana *et al.* 2015] Surender Baswana, Manoj Gupta and Sandeep Sen. *Fully Dynamic Maximal Matching in $O(\log(n)$ Update Time.* SIAM Journal on Computing, vol. 44, no. 1, pages 88–113, 2015. (Cited on page 69.)

[Bazgan *et al.* 2015] Cristina Bazgan, Florian Jamain and Daniel Vanderpooten. *Approximate Pareto sets of minimal size for multi-objective optimization problems.* Operations Research Letters, vol. 43, no. 1, pages 1–6, 2015. (Cited on page 3.)

[Ben-Or & Tiwari 1988] Michael Ben-Or and Prasoon Tiwari. *A deterministic algorithm for sparse multivariate polynomial interpolation.* In Proceedings of the twentieth annual ACM symposium on Theory of computing, pages 301–309, 1988. (Cited on page 59.)

[Bérczi *et al.* 2021] Kristóf Bérczi, Endre Boros, Ondřej Čepek, Khaled Elbassioni, Petr Kučera and Kazuhisa Makino. *Generating clause sequences of a CNF formula.* Theoretical Computer Science, vol. 856, pages 68–74, 2021. (Cited on pages 15 and 73.)

[Berkholz *et al.* 2020] Christoph Berkholz, Fabian Gerhardt and Nicole Schweikardt. *Constant delay enumeration for conjunctive queries: a tutorial.* ACM SIGLOG News, vol. 7, no. 1, pages 4–33, 2020. (Cited on page 57.)

[Blind *et al.* 2020] Sarah Blind, Nadia Creignou and Frédéric Olive. *Locally definable vertex set properties are efficiently enumerable.* Discrete Applied Mathematics, 2020. (Cited on page 33.)

[Böhler *et al.* 2005] Elmar Böhler, Steffen Reith, Henning Schnoor and Heribert Vollmer. *Bases for Boolean co-clones.* Information Processing Letters, vol. 96, no. 2, pages 59–66, 2005. (Cited on page 54.)

[Böhmová *et al.* 2018] 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, vol. 62, no. 3, pages 600–621, 2018. (Cited on page 3.)

[Bonamy *et al.* 2019] 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. (Cited on page 23.)

[Bonamy *et al.* 2020] Marthe Bonamy, Oscar Defrain, Marc Heinrich, Michał Pilipczuk and Jean-Florent Raymond. *Enumerating minimal dominating sets in Kt-free graphs and variants.* ACM Transactions on Algorithms (TALG), vol. 16, no. 3, pages 1–23, 2020. (Cited on page 16.)

[Boros *et al.* 1998] Endre Boros, Vladimir Gurvich and Peter L Hammer. *Dual subimplicants of positive Boolean functions.* Optimization Methods and Software, vol. 10, no. 2, pages 147–156, 1998. (Cited on page 49.)

[Boros *et al.* 2009] 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), vol. 49, pages 15–43, 2009. (Cited on page 48.)

[Brault-Baron 2013] Johann Brault-Baron. *De la pertinence de l'énumération: complexité en logiques propositionnelle et du premier ordre.* PhD thesis, 2013. (Cited on pages 1, 9 and 57.)

[Bron & Kerbosch 1973] Coen Bron and Joep Kerbosch. *Algorithm 457: finding all cliques of an undirected graph.* Communications of the ACM, vol. 16, no. 9, pages 575–577, 1973. (Cited on pages 13 and 49.)

[Bulatov *et al.* 2012] Andrei A Bulatov, Víctor Dalmau, Martin Grohe and Dániel Marx. *Enumerating homomorphisms.* Journal of Computer and System Sciences, vol. 78, no. 2, pages 638–650, 2012. (Cited on pages 4 and 53.)

[Bulatov *et al.* 2016] Andrei Bulatov, Marcin Kozik, Peter Mayr and Markus Steindl. *The subpower membership problem for semigroups.* International Journal of Algebra and Computation, vol. 26, no. 07, pages 1435–1451, 2016. (Cited on page 55.)

[Bulatov 2017] Andrei A Bulatov. *A dichotomy theorem for nonuniform CSPs.* In 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS), pages 319–330. IEEE, 2017. (Cited on pages 52 and 53.)

[Cao 2020] Yixin Cao. *Enumerating maximal induced subgraphs.* arXiv preprint arXiv:2004.09885, 2020. (Cited on pages 20 and 50.)

[Capelli & Strozecki 2019] Florent Capelli and Yann Strozecki. *Incremental delay enumeration: Space and time.* Discrete Applied Mathematics, vol. 268, pages 179–190, 2019. (Cited on pages 1, 14, 15, 16, 17, 20, 28, 30 and 40.)

[Capelli & Strozecki 2021a] Florent Capelli and Yann Strozecki. *Enumerating models of DNF faster: Breaking the dependency on the formula size.* Discrete Applied Mathematics, vol. 303, pages 203–215, 2021. Combined Special Issue: 1) 17th Cologne–Twente Workshop on Graphs and Combinatorial Optimization (CTW 2019); Guest edited by Johann Hurink, Bodo Manthey 2) WEPA 2018 (Second Workshop on Enumeration Problems and Applications); Guest edited by Takeaki Uno, Andrea Marino. (Cited on pages 1, 34, 35, 38, 48, 58, 70 and 73.)

[Capelli & Strozecki 2021b] Florent Capelli and Yann Strozecki. Regularizing enumeration delay with polynomial memory. 2021. (Cited on pages 28, 29, 31, 32, 34, 35 and 48.)

[Carmeli & Kröll 2019] 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. (Cited on page 16.)

[Carmeli *et al.* 2020] Nofar Carmeli, Shai Zeevi, Christoph Berkholz, Benny Kimelfeld and Nicole Schweikardt. *Answering (Unions of) Conjunctive Queries using Random Access and Random-Order Enumeration.* In Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, pages 393–409, 2020. (Cited on page 45.)

[Carmeli *et al.* 2021] Nofar Carmeli, Nikolaos Tziavelis, Wolfgang Gatterbauer, Benny Kimelfeld and Mirek Riedewald. *Tractable Orders for Direct Access to Ranked Answers of Conjunctive Queries.* In Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, pages 325–341, 2021. (Cited on page 45.)

[Casel *et al.* 2019a] 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. (Cited on page 49.)

[Casel *et al.* 2019b] 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. (Cited on page 49.)

[Cazals & Karande 2008] Frédéric Cazals and Chinmay Karande. *A note on the problem of reporting maximal cliques*. Theoretical Computer Science, vol. 407, no. 1-3, pages 564–568, 2008. (Cited on page 63.)

[Chan 1996] Timothy M Chan. *Optimal output-sensitive convex hull algorithms in two and three dimensions*. Discrete & Computational Geometry, vol. 16, no. 4, pages 361–368, 1996. (Cited on page 3.)

[Chattopadhyay *et al.* 2021] Arkadev Chattopadhyay, Bruno Grenet, Pascal Koiran, Natacha Portier and Yann Strozecki. *Computing the multilinear factors of lacunary polynomials without heights*. Journal of Symbolic Computation, vol. 104, pages 183–206, 2021. (Cited on page 71.)

[Cohen *et al.* 2008] 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, vol. 74, no. 7, pages 1147–1159, 2008. (Cited on pages 4 and 50.)

[Conte & Tomita 2021] Alessio Conte and Etsuji Tomita. *Overall and Delay Complexity of the CLIQUES and Bron-Kerbosch Algorithms*. In WALCOM: Algorithms and Computation: 15th International Conference and Workshops, WALCOM 2021, Yangon, Myanmar, February 28–March 2, 2021, Proceedings 15, pages 195–207. Springer International Publishing, 2021. (Cited on page 49.)

[Conte & Uno 2019] 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. (Cited on pages 4 and 20.)

[Conte *et al.* 2016] 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. (Cited on pages 25 and 49.)

[Conte *et al.* 2018] 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. (Cited on page 70.)

[Conte *et al.* 2019] Alessio Conte, Roberto Grossi, Andrea Marino and Luca Versari. *Listing Maximal Subgraphs Satisfying Strongly Accessible Properties*. SIAM Journal on Discrete Mathematics, vol. 33, no. 2, pages 587–613, 2019. (Cited on pages 4, 25, 48 and 50.)

[Cook & Rackoff 1980] Stephen A Cook and Charles W Rackoff. *Space lower bounds for maze threadability on restricted machines*. SIAM Journal on Computing, vol. 9, no. 3, pages 636–652, 1980. (Cited on page 51.)

[Cook & Reckhow 1973] Stephen A. Cook and Robert A. Reckhow. *Time Bounded Random Access Machines*. J. Comput. Syst. Sci., vol. 7, no. 4, pages 354–375, 1973. (Cited on pages 8, 9 and 10.)

[Cook 1971] 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. (Cited on page 11.)

[Cormen *et al.* 2009] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest and Clifford Stein. Introduction to algorithms. MIT press, 2009. (Cited on page 9.)

[Courcelle 2009] Bruno Courcelle. *Linear delay enumeration and monadic second-order logic*. Discret. Appl. Math., vol. 157, no. 12, pages 2675–2700, 2009. (Cited on pages 33, 45 and 57.)

[Creignou & Hébrard 1997] Nadia Creignou and Jean-Jacques Hébrard. *On generating all solutions of generalized satisfiability problems*. Informatique théorique et applications, vol. 31, no. 6, pages 499–511, 1997. (Cited on pages 4, 22 and 53.)

[Creignou & Vollmer 2015] Nadia Creignou and Heribert Vollmer. *Parameterized complexity of weighted satisfiability problems: Decision, enumeration, counting*. Fundamenta Informaticae, vol. 136, no. 4, pages 297–316, 2015. (Cited on page 69.)

[Creignou *et al.* 2001] Nadia Creignou, Sanjeev Khanna and Madhu Sudan. Complexity classifications of boolean constraint satisfaction problems. SIAM, 2001. (Cited on page 22.)

[Creignou *et al.* 2011] Nadia Creignou, Frédéric Olive and Johannes Schmidt. *Enumerating All Solutions of a Boolean CSP by Non-decreasing Weight*. In Karem A. Sakallah and Laurent Simon, editors, Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings, volume 6695 of *Lecture Notes in Computer Science*, pages 120–133. Springer, 2011. (Cited on pages 44 and 53.)

[Creignou *et al.* 2017a] 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. (Cited on pages 8, 15, 23 and 25.)

[Creignou *et al.* 2017b] Nadia Creignou, Arne Meier, Julian-Steffen Müller, Johannes Schmidt and Heribert Vollmer. *Paradigms for parameterized enumeration.* Theory of Computing Systems, vol. 60, no. 4, pages 737–758, 2017. (Cited on page 69.)

[Daigo & Hirata 2009] Taishin Daigo and Kouichi Hirata. *On generating all maximal acyclic subhypergraphs with polynomial delay.* In International Conference on Current Trends in Theory and Practice of Computer Science, pages 181–192. Springer, 2009. (Cited on page 44.)

[Darwiche & Marquis 2002] Adnan Darwiche and Pierre Marquis. *A knowledge compilation map.* Journal of Artificial Intelligence Research, vol. 17, pages 229–264, 2002. (Cited on page 71.)

[de Colnet & Mengel 2020] Alexis de Colnet and Stefan Mengel. *Lower Bounds for Approximate Knowledge Compilation.* In Christian Bessiere, editor, Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020, pages 1834–1840. ijcai.org, 2020. (Cited on page 73.)

[Defrain & Nourine 2019] Oscar Defrain and Lhouari Nourine. *Dualization in lattices given by implicational bases.* In International Conference on Formal Concept Analysis, pages 89–98. Springer, 2019. (Cited on pages 47 and 48.)

[Defrain *et al.* 2021] Oscar Defrain, Lhouari Nourine and Takeaki Uno. *On the dualization in distributive lattices and related problems.* Discrete Applied Mathematics, vol. 300, pages 85–96, 2021. (Cited on page 47.)

[Disser & Skutella 2018] Yann Disser and Martin Skutella. *The simplex algorithm is NP-mighty.* ACM Transactions on Algorithms (TALG), vol. 15, no. 1, pages 1–19, 2018. (Cited on page 49.)

[Duchon *et al.* 2004] Philippe Duchon, Philippe Flajolet, Guy Louchard and Gilles Schaeffer. *Boltzmann samplers for the random generation of combinatorial structures.* Combinatorics, Probability and Computing, vol. 13, no. 4-5, pages 577–625, 2004. (Cited on page 41.)

[Durand & Flajolet 2003] Marianne Durand and Philippe Flajolet. *Loglog counting of large cardinalities.* In European Symposium on Algorithms, pages 605–617. Springer, 2003. (Cited on page 72.)

[Durand & Grandjean 2007] Arnaud Durand and Etienne Grandjean. *First-order queries on structures of bounded degree are computable with constant delay.* ACM Trans. Comput. Log., vol. 8, no. 4, page 21, 2007. (Cited on pages 10, 39, 56 and 72.)

[Durand & Strozecki 2011] 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. (Cited on pages 33, 57 and 70.)

[Durand *et al.* 2005] Arnaud Durand, Miki Hermann and Phokion G Kolaitis. *Subtractive reductions and complete problems for counting complexity classes.* Theoretical Computer Science, vol. 340, no. 3, pages 496–513, 2005. (Cited on page 21.)

[Durand *et al.* 2020] Arnaud Durand, Nicole Schweikardt and Luc Segoufin. *Enumerating Answers to First-Order Queries over Databases of Low Degree.* CoRR, vol. abs/2010.08382, 2020. (Cited on page 56.)

[Durand 2020] Arnaud Durand. *Fine-Grained Complexity Analysis of Queries: From Decision to Counting and Enumeration.* In Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, pages 331–346, 2020. (Cited on page 39.)

[Ehrlich & Rarey 2011] 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, vol. 1, no. 1, pages 68–79, 2011. (Cited on page 3.)

[Eiter & Gottlob 1995] Thomas Eiter and Georg Gottlob. *Identifying the minimal transversals of a hypergraph and related problems.* SIAM Journal on Computing, vol. 24, no. 6, pages 1278–1304, 1995. (Cited on pages 48 and 56.)

[Eiter *et al.* 2003] Thomas Eiter, Georg Gottlob and Kazuhisa Makino. *New results on monotone dualization and generating hypergraph transversals.* SIAM Journal on Computing, vol. 32, no. 2, pages 514–537, 2003. (Cited on page 3.)

[Eppstein *et al.* 2015] David Eppstein *et al. K-Best Enumeration.* Bulletin of EATCS, vol. 1, no. 115, 2015. (Cited on pages 14 and 43.)

[Erdos & Rényi 1961] Paul Erdos and Alfred Rényi. *On a classical problem of probability theory.* Magyar Tud. Akad. Mat. Kutató Int. Közl, vol. 6, no. 1-2, pages 215–220, 1961. (Cited on page 40.)

[Feder 1994] Tomás Feder. *Network flow and 2-satisfiability.* Algorithmica, vol. 11, no. 3, pages 291–319, 1994. (Cited on pages 38 and 54.)

[Flajolet & Martin 1985] Philippe Flajolet and G Nigel Martin. *Probabilistic counting algorithms for data base applications.* Journal of computer and system sciences, vol. 31, no. 2, pages 182–209, 1985. (Cited on page 41.)

[Florêncio *et al.* 2015] 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, vol. 21, no. 7, pages 891–911, 2015. (Cited on pages 7 and 33.)

[Fomin & Kratsch 2010] Fedor V Fomin and Dieter Kratsch. Exact exponential algorithms. Springer Science & Business Media, 2010. (Cited on page 3.)

[Fournier *et al.* 2015] Hervé Fournier, Guillaume Malod and Stefan Mengel. *Monomials in arithmetic circuits: Complete problems in the counting hierarchy.* computational complexity, vol. 24, no. 1, pages 1–30, 2015. (Cited on page 70.)

[Fredkin 1960] Edward Fredkin. *Trie memory.* Communications of the ACM, vol. 3, no. 9, pages 490–499, 1960. (Cited on page 34.)

[Fredman & Khachiyan 1996] Michael L Fredman and Leonid Khachiyan. *On the complexity of dualization of monotone disjunctive normal forms.* Journal of Algorithms, vol. 21, no. 3, pages 618–628, 1996. (Cited on page 3.)

[Fredman & Willard 1993] Michael L Fredman and Dan E Willard. *Surpassing the information theoretic bound with fusion trees.* Journal of computer and system sciences, vol. 47, no. 3, pages 424–436, 1993. (Cited on page 10.)

[Frick & Grohe 2004] Markus Frick and Martin Grohe. *The complexity of first-order and monadic second-order logic revisited.* Annals of pure and applied logic, vol. 130, no. 1-3, pages 3–31, 2004. (Cited on page 57.)

[Friedmann *et al.* 2011] Oliver Friedmann, Thomas Dueholm Hansen and Uri Zwick. *Subexponential lower bounds for randomized pivoting rules for the simplex algorithm.* In Proceedings of the forty-third annual ACM symposium on Theory of computing, pages 283–292, 2011. (Cited on page 49.)

[Goldberg 1991] Leslie Ann Goldberg. *Efficient algorithms for listing combinatorial structures.* PhD thesis, University of Edinburgh, UK, 1991. (Cited on pages 40, 50 and 72.)

[Golovach *et al.* 2018] 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, vol. 80, no. 2, pages 714–741, 2018. (Cited on page 4.)

[Golovach *et al.* 2021] Petr A Golovach, Christian Komusiewicz, Dieter Kratsch and Van Bang Le. *Refined Notions of Parameterized Enumeration Kernels with Applications to Matching Cut Enumeration.* arXiv preprint arXiv:2101.03800, 2021. (Cited on page 69.)

[Grandjean & Schwentick 2002] Etienne Grandjean and Thomas Schwentick. *Machine-independent characterizations and complete problems for deterministic linear time.* SIAM Journal on Computing, vol. 32, no. 1, pages 196–230, 2002. (Cited on page 10.)

[Greco & Scarcello 2013] Gianluigi Greco and Francesco Scarcello. *Structural tractability of enumerating CSP solutions.* Constraints, vol. 18, no. 1, pages 38–74, 2013. (Cited on page 53.)

[Guiraud 2021] Maël Guiraud. *Deterministic scheduling of periodic datagrams for low latency in 5G and beyond.* PhD thesis, Université Paris Saclay, 2021. (Cited on page 71.)

[Hagen 2008] Matthias Hagen. " algorithmic and computational complexity issues of monet. Cuvillier Verlag, 2008. (Cited on pages 2, 7 and 48.)

[Hanaka *et al.* 2021] Tesshu Hanaka, Yasuaki Kobayashi, Kazuhiro Kurita and Yota Otachi. *Finding Diverse Trees, Paths, and More*. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 35, pages 3778–3786, 2021. (Cited on page 74.)

[Hartmanis & Stearns 1965] Juris Hartmanis and Richard E Stearns. *On the computational complexity of algorithms*. Transactions of the American Mathematical Society, vol. 117, pages 285–306, 1965. (Cited on pages 21 and 37.)

[Hartung *et al.* 2019] Elizabeth Hartung, Hung Phuc Hoang, Torsten Mütze and Aaron Williams. *Combinatorial generation via permutation languages. I. Fundamentals*. arXiv preprint arXiv:1906.06069, 2019. (Cited on page 38.)

[Hastad 1986] 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. (Cited on page 52.)

[Holton & McKay 1988] Derek A Holton and Brendan D McKay. *The smallest non-hamiltonian 3-connected cubic planar graphs have 38 vertices*. Journal of Combinatorial Theory, Series B, vol. 45, no. 3, pages 305–319, 1988. (Cited on page 66.)

[Hopcroft & Tarjan 1973] John E. Hopcroft and Robert Endre Tarjan. *A V log V algorithm for isomorphism of triconnected planar graphs*. Journal of Computer and System Sciences, vol. 7, no. 3, pages 323–331, 1973. (Cited on page 66.)

[Hopcroft & Wong 1974] John E Hopcroft and Jin-Kue Wong. *Linear time algorithm for isomorphism of planar graphs (preliminary report)*. In ACM symposium on Theory of computing, pages 172–184, 1974. (Cited on page 66.)

[Jerrum 2003] Mark Jerrum. Counting, sampling and integrating: algorithms and complexity. Springer Science & Business Media, 2003. (Cited on pages 40 and 58.)

[Jiang *et al.* 2013] Chuntao Jiang, Frans Coenen and Michele Zito. *A survey of frequent subgraph mining algorithms*. The Knowledge Engineering Review, vol. 28, no. 1, pages 75–105, 2013. (Cited on page 2.)

[Johnson *et al.* 1988] David S. Johnson, Christos H. Papadimitriou and Mihalis Yannakakis. *On Generating All Maximal Independent Sets*. Inf. Process. Lett., vol. 27, no. 3, pages 119–123, 1988. (Cited on pages 2, 27, 43 and 44.)

[Kabanets & Impagliazzo 2004] Valentine Kabanets and Russell Impagliazzo. *Derandomizing polynomial identity tests means proving circuit lower bounds*. computational complexity, vol. 13, no. 1-2, pages 1–46, 2004. (Cited on page 61.)

[Kane *et al.* 2010] Daniel M Kane, Jelani Nelson and David P Woodruff. *An optimal algorithm for the distinct elements problem*. In Proceedings of the twenty-ninth ACM

SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pages 41–52. ACM, 2010. (Cited on page 41.)

[Kanté *et al.* 2014] 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, vol. 28, no. 4, pages 1916–1929, 2014. (Cited on pages 23 and 48.)

[Kanté *et al.* 2015] 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. (Cited on page 23.)

[Karmarkar 1984] Narendra Karmarkar. *A new polynomial-time algorithm for linear programming*. In Proceedings of the sixteenth annual ACM symposium on Theory of computing, pages 302–311, 1984. (Cited on page 49.)

[Kavvadias *et al.* 2000] Dimitris J Kavvadias, Martha Sideri and Elias C Stavropoulos. *Generating all maximal models of a Boolean expression*. Information Processing Letters, vol. 74, no. 3-4, pages 157–162, 2000. (Cited on pages 47 and 56.)

[Kawahara *et al.* 2019] Jun Kawahara, Toshiki Saitoh, Hirofumi Suzuki and Ryo Yoshinaka. *Colorful Frontier-Based Search: Implicit Enumeration of Chordal and Interval Subgraphs*. In International Symposium on Experimental Algorithms, pages 125–141. Springer, 2019. (Cited on page 71.)

[Kazana & Segoufin 2020] Wojciech Kazana and Luc Segoufin. *First-order queries on classes of structures with bounded expansion*. Log. Methods Comput. Sci., vol. 16, no. 1, 2020. (Cited on page 56.)

[Khachiyan *et al.* 2005] 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, vol. 19, no. 4, pages 966–984, 2005. (Cited on pages 3, 15, 27 and 56.)

[Khachiyan *et al.* 2006] 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. (Cited on page 47.)

[Kiefer *et al.* 2011] S. Kiefer, A. Murawski, J. Ouaknine, B. Wachter and J. Worrell. *Language Equivalence for Probabilistic Automata*. In Computer Aided Verification, pages 526–540. Springer, 2011. (Cited on page 58.)

[Klivans & Spielman 2001] Adam R Klivans and Daniel Spielman. *Randomness efficient identity testing of multivariate polynomials*. In Proceedings of the thirty-third annual ACM symposium on Theory of computing, pages 216–223, 2001. (Cited on pages 59 and 60.)

[Knuth 1997] Donald Ervin Knuth. The art of computer programming, volume 3. Pearson Education, 1997. (Cited on page 3.)

[Knuth 2011] Donald E Knuth. The art of computer programming, volume 4a: combinatorial algorithms, part 1. Pearson Education India, 2011. (Cited on pages 2 and 38.)

[Kobayashi *et al.* 2020] Yasuaki Kobayashi, Kazuhiro Kurita and Kunihiro Wasa. *Efficient Constant-Factor Approximate Enumeration of Minimal Subsets for Monotone Properties with Cardinality Constraints.* arXiv preprint arXiv:2009.08830, 2020. (Cited on page 72.)

[Kopelowitz *et al.* 2016] 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. (Cited on page 48.)

[Kreutzer 2009] Stephan Kreutzer. *On the parameterised intractability of monadic second-order logic.* In International Workshop on Computer Science Logic, pages 348–363. Springer, 2009. (Cited on page 57.)

[Ladner 1975] Richard E Ladner. *On the structure of polynomial time reducibility.* Journal of the ACM (JACM), vol. 22, no. 1, pages 155–171, 1975. (Cited on page 52.)

[Lawler *et al.* 1980] 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., vol. 9, no. 3, pages 558–565, 1980. (Cited on pages 18, 25, 43 and 50.)

[Li & Ruskey 1999] Gang Li and Frank Ruskey. *The advantages of forward thinking in generating rooted and free trees.* In ACM-SIAM symposium on Discrete algorithms, pages 939–940, 1999. (Cited on page 64.)

[Lovász & Simonovits 1993] László Lovász and Miklós Simonovits. *Random walks in a convex body and an improved volume algorithm.* Random structures & algorithms, vol. 4, no. 4, pages 359–412, 1993. (Cited on page 40.)

[Lucas 1882] 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. (Cited on page 2.)

[Marino 2015] Andrea Marino. *Enumeration algorithms.* In Analysis and Enumeration, pages 13–35. Springer, 2015. (Cited on page 1.)

[Mary & Strozecki 2016a] Arnaud Mary and Yann Strozecki. *Efficient Enumeration of Solutions Produced by Closure Operations.* In 33rd Symposium on Theoretical Aspects of Computer Science, 2016. (Cited on pages 15, 53 and 55.)

[Mary & Strozecki 2016b] Arnaud Mary and Yann Strozecki. *Generating maximal solutions given by closure operations.* First Workshop on Enumeration Problems and Applications, 2016. (Cited on page 53.)

[Mary & Strozecki 2019] Arnaud Mary and Yann Strozecki. *Efficient enumeration of solutions produced by closure operations*. Discrete Mathematics & Theoretical Computer Science, vol. 21, no. 3, 2019. (Cited on pages 1, 4, 18, 35, 48, 53 and 70.)

[Mary 2013] Arnaud Mary. *Énumération des Dominants Minimaux d'un graphe*. PhD thesis, Université Blaise Pascal, 2013. (Cited on pages 1, 4, 9 and 22.)

[Masbaum & Vaintrob 2002] Gregor Masbaum and Arkady Vaintrob. *A new matrix-tree theorem*. International Mathematics Research Notices, vol. 2002, no. 27, pages 1397–1426, 2002. (Cited on page 58.)

[McGregor 2014] Andrew McGregor. *Graph stream algorithms: a survey*. ACM SIGMOD Record, vol. 43, no. 1, pages 9–20, 2014. (Cited on page 72.)

[Megiddo & Papadimitriou 1991] Nimrod Megiddo and Christos H Papadimitriou. *On total functions, existence theorems and computational complexity*. Theoretical Computer Science, vol. 81, no. 2, pages 317–324, 1991. (Cited on page 16.)

[Meier 2020] Arne Meier. *Incremental FPT Delay*. Algorithms, vol. 13, no. 5, page 122, 2020. (Cited on page 69.)

[Murakami & Uno 2014] Keisuke Murakami and Takeaki Uno. *Efficient algorithms for dualizing large-scale hypergraphs*. Discrete Applied Mathematics, vol. 170, pages 83–94, 2014. (Cited on page 72.)

[Nakano & Uno 2003] Shin-ichi Nakano and Takeaki Uno. *A simple constant time enumeration algorithm for free trees*. PSJ SIGNotes ALgorithms, no. 091-002, 2003. (Cited on page 38.)

[Nishimura 2018] Naomi Nishimura. *Introduction to reconfiguration*. Algorithms, vol. 11, no. 4, page 52, 2018. (Cited on page 41.)

[Olteanu & Závodný 2012] Dan Olteanu and Jakub Závodný. *Factorised representations of query results: Size bounds and readability*. In Proceedings of the 15th International Conference on Database Theory, pages 285–298, 2012. (Cited on page 71.)

[Papadimitriou & Yannakakis 2000] 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. (Cited on pages 3 and 72.)

[Papadimitriou 2003] Christos H Papadimitriou. Computational complexity. John Wiley and Sons Ltd., 2003. (Cited on page 9.)

[Park & Phillips 1993] James K Park and Cynthia A Phillips. *Finding minimum-quotient cuts in planar graphs*. In ACM symposium on Theory of computing, pages 766–775, 1993. (Cited on page 65.)

[Peyronnet *et al.* 2012] S. Peyronnet, M. De Rougemont and Y. Strozecki. *Approximate verification and enumeration problems*. Theoretical Aspects of Computing–ICTAC 2012, pages 228–242, 2012. (Cited on pages 40 and 58.)

[Post 1941] Emil Leon Post. The two-valued iterative systems of mathematical logic. Princeton University Press, 1941. (Cited on page 54.)

[Pruesse & Ruskey 1994] Gara Pruesse and Frank Ruskey. *Generating linear extensions fast*. SIAM Journal on Computing, vol. 23, no. 2, pages 373–386, 1994. (Cited on page 3.)

[Razborov 1985] Alexander A Razborov. *Lower bounds for the monotone complexity of some Boolean functions*. In Soviet Math. Dokl., volume 31, pages 354–357, 1985. (Cited on page 52.)

[Read & Tarjan 1975] RC Read and RE Tarjan. *Bounds on backtrack algorithms for listing cycles, paths, and spanning trees*. Networks, vol. 5, no. 3, pages 237–252, 1975. (Cited on page 4.)

[Reith & Vollmer 2003] Steffen Reith and Heribert Vollmer. *Optimal satisfiability for propositional calculi and constraint satisfaction problems*. Information and Computation, vol. 186, no. 1, pages 1–19, 2003. (Cited on page 54.)

[Ruskey 2003] Frank Ruskey. *Combinatorial generation*. Preliminary working draft. University of Victoria, Victoria, BC, Canada, vol. 11, page 20, 2003. (Cited on pages 2, 38 and 71.)

[Saluja *et al.* 1995] Sanjeev Saluja, KV Subrahmanyam and Madhukar N Thakur. *Descriptive complexity of# P functions*. Journal of Computer and System Sciences, vol. 50, no. 3, pages 493–505, 1995. (Cited on page 57.)

[Savickỳ & Kučera 2016] Petr Savickỳ and Petr Kučera. *Generating models of a matched formula with a polynomial delay*. Journal of Artificial Intelligence Research, vol. 56, pages 379–402, 2016. (Cited on page 49.)

[Schaefer 1978] Thomas J Schaefer. *The complexity of satisfiability problems*. In Proceedings of the tenth annual ACM symposium on Theory of computing, pages 216–226, 1978. (Cited on page 52.)

[Schmidt 2017] Johannes Schmidt. *The Weight in Enumeration*. In Frank Drewes, Carlos Martín-Vide and Bianca Truthe, editors, Language and Automata Theory and Applications - 11th International Conference, LATA 2017, Umeå, Sweden, March 6-9, 2017, Proceedings, volume 10168 of *Lecture Notes in Computer Science*, pages 208–219, 2017. (Cited on pages 44 and 53.)

[Schnoor & Schnoor 2007] Henning Schnoor and Ilka Schnoor. *Enumerating all solutions for constraint satisfaction problems*. In Annual Symposium on Theoretical Aspects of Computer Science, pages 694–705. Springer, 2007. (Cited on page 53.)

[Schweikardt *et al.* 2018] Nicole Schweikardt, Luc Segoufin and Alexandre Vigny. *Enumeration for FO Queries over Nowhere Dense Graphs*. In Jan Van den Bussche and Marcelo Arenas, editors, Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018, pages 151–163. ACM, 2018. (Cited on pages 26 and 56.)

[Segoufin 2015] Luc Segoufin. *Constant Delay Enumeration for Conjunctive Queries*. SIGMOD Record, vol. 44, no. 1, pages 10–17, 2015. (Cited on pages 4 and 39.)

[Staub 2016] Ruben Staub. Calculs d'isomorphismes pour la chemoinformatique. Master's thesis, Université de Versailles Saint-Quentin-en-Yvelines, 2016. (Cited on page 66.)

[Steindl 2017] Markus Steindl. *The subpower membership problem for bands*. Journal of Algebra, vol. 489, pages 529–551, 2017. (Cited on page 55.)

[Strozecki 2010a] Y. Strozecki. *Enumeration of the monomials of a polynomial and related complexity classes*. Mathematical Foundations of Computer Science, pages 629–640, 2010. (Cited on pages 13 and 59.)

[Strozecki 2010b] Yann Strozecki. *Enumeration complexity and matroid decomposition*. PhD thesis, Université Paris Diderot - Paris 7, 2010. (Cited on pages 1, 8, 9, 15, 22, 26 and 44.)

[Strozecki 2011] Yann Strozecki. *Monadic second-order model-checking on decomposable matroids*. Discrete Applied Mathematics, vol. 159, no. 10, pages 1022–1039, 2011. (Cited on page 57.)

[Strozecki 2013] Yann Strozecki. *On enumerating monomials and other combinatorial structures by polynomial interpolation*. Theory of Computing Systems, vol. 53, no. 4, pages 532–568, 2013. (Cited on pages 59, 60 and 61.)

[Strozecki 2019] Yann Strozecki. *Enumeration Complexity*. Bulletin of EATCS, vol. 1, no. 129, 2019. (Cited on page 1.)

[Tiernan 1970] James C Tiernan. *An efficient search algorithm to find the elementary circuits of a graph*. Communications of the ACM, vol. 13, no. 12, pages 722–726, 1970. (Cited on page 2.)

[Tomita *et al.* 2006] Etsuji Tomita, Akira Tanaka and Haruhisa Takahashi. *The worst-case time complexity for generating all maximal cliques and computational experiments*. Theoretical computer science, vol. 363, no. 1, pages 28–42, 2006. (Cited on page 49.)

[Uno 1997] Takeaki Uno. *Algorithms for enumerating all perfect, maximum and maximal matchings in bipartite graphs*. In International Symposium on Algorithms and Computation, pages 92–101. Springer, 1997. (Cited on pages 26 and 45.)

[Uno 2015] Takeaki Uno. *Constant time enumeration by amortization*. In Workshop on Algorithms and Data Structures, pages 593–605. Springer, 2015. (Cited on pages 3, 34 and 38.)

[Valiant 1979] Leslie G Valiant. *The complexity of computing the permanent.* Theoretical computer science, vol. 8, no. 2, pages 189–201, 1979. (Cited on page 45.)

[Vassilvitskii & Yannakakis 2005] Sergei Vassilvitskii and Mihalis Yannakakis. *Efficiently computing succinct trade-off curves.* Theoretical Computer Science, vol. 348, no. 2-3, pages 334–356, 2005. (Cited on page 3.)

[Vigny 2018] Alexandre Vigny. *Query enumeration and nowhere dense graphs.* PhD thesis, 2018. (Cited on pages 1, 51 and 57.)

[Walsh 2003] 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. (Cited on page 38.)

[Wang *et al.* 2020] Yishu Wang, Arnaud Mary, Marie-France Sagot and Blerina Sinaimeri. *Lazy listing of equivalence classes–A paper on dynamic programming and tropical circuits.* arXiv preprint arXiv:2004.12143, 2020. (Cited on page 71.)

[Wasa 2016] Kunihiro Wasa. *Enumeration of enumeration algorithms.* arXiv preprint arXiv:1605.05102, 2016. (Cited on page 2.)

[Wigderson 1992] Avi Wigderson. *The complexity of graph connectivity.* In International Symposium on Mathematical Foundations of Computer Science, pages 112–132. Springer, 1992. (Cited on page 51.)

[Wild 2012] Marcel Wild. *Compactly generating all satisfying truth assignments of a Horn formula.* Journal on satisfiability, Boolean modeling and computation, vol. 8, no. 1-2, pages 63–82, 2012. (Cited on page 70.)

[Williams 2014] Ryan Williams. *Nonuniform ACC circuit lower bounds.* Journal of the ACM (JACM), vol. 61, no. 1, page 2, 2014. (Cited on page 52.)

[Williams 2018] Virginia Vassilevska Williams. *On some fine-grained questions in algorithms and complexity.* In Proceedings of the ICM, 2018. (Cited on page 16.)

[Wright *et al.* 1986] Robert Alan Wright, Bruce Richmond, Andrew Odlyzko and Brendan D McKay. *Constant time generation of free trees.* SIAM Journal on Computing, vol. 15, no. 2, pages 540–548, 1986. (Cited on pages 3 and 38.)

[Zhuk 2020] Dmitriy Zhuk. *A Proof of the CSP Dichotomy Conjecture.* Journal of the ACM (JACM), vol. 67, no. 5, pages 1–78, 2020. (Cited on pages 52 and 53.)

[Zippel 1979] Richard Zippel. *Probabilistic algorithms for sparse polynomials.* In International symposium on symbolic and algebraic manipulation, pages 216–226. Springer, 1979. (Cited on page 59.)

[Zippel 1990] R. Zippel. *Interpolating polynomials from their values.* Journal of Symbolic Computation, vol. 9, no. 3, pages 375–403, 1990. (Cited on pages 13, 59 and 61.)

Appendices

Curriculum Vitae

Find in this section my current CV.

Yann Strozecki

CV

45, avenue des États-Unis
78035 Versailles
✉ yann.strozecki@uvsq.fr
🖥 https://yann-strozecki.github.io/
38 years old, 2 children

Academic Appointments

2012– **Maître de conférences (Associate professor)**, *Versailles University (UVSQ)*.
Member of the ALMOST team, DAVID laboratory

2017–2018 **Délégation CNRS (Visiting professor)**, *Sorbonne University*.
Member of the RO team, LIP6

2011–2012 **ATER (Assistant professor)**, *Paris-Sud University* .
Member of the ALGO team, LRI

2011 **Postdoctoral student**, *University of Toronto*.
Member of the Theory Group

2010–2011 **Part-time ATER (Assistant professor)**, *Paris Diderot University*.
Member of the Équipe Logique Mathématique, IMJ

Responsabilities

2021– **Board member of HYPHES**, *University of Versailles and DCbrain*, HYPHES is a joint laboratory between DAVID an academic laboratory and DCbrain an innovative start-up.

2020– **Head of the ALMOST team**, *Versailles University (UVSQ)*, DAVID Laboratory, Responsible of ALMOST, **Al**gorithmique et **Mo**dèle **St**ochastique, one of the four teams of DAVID laboratory. The team currently has 11 permanent members (assistant professors and full professors), 3 associate members and 12 PhD students.

2015– **Board member of DAVID Laboratory**, *Versailles University (UVSQ)*.

2013–2015 **Elected to the laboratory council of the PRISM**, *Versailles University (UVSQ)*.

2008–2010 **Representative of the PhD students in the Équipe de Logique Mathématique**, *University Paris-Diderot*.

Education

2007–2010 **PhD Thesis (supervisor: Arnaud Durand)**, *Paris Diderot University*, Équipe Logique Mathématique, IMJ.
Title: *Complexity of enumeration and matroid decomposition*
Examiners: Bruno Courcelle, Daniel Král', Michel De Rougemont
Jury: Arnaud Durand, Miki Hermann, Guillaume Malod, Gilles Villard

2006–2007 **Master in mathematics**, *University Paris Diderot*.
Master of Logic for Mathematics and Foundation of Computer Science.
Internship with Arnaud Durand on holographic algorithms.

2003–2006 **Élève normalien**, *ENS Lyon*.
Bachelor of mathematics, Master 1 in computer science, Agrégation in mathematics with specialisation in computer science

Research activities

I am interested in *complexity* and *algorithmic*, specifically of enumeration problems, that is the generation of all solutions of a problem. I am also interested in algorithmic game theory, graph algorithms (with applications to networks, cheminformatic . . .), graph decomposition and logic.

Scientific Animation:
- Talks (about 100) in seminars, workshops and conferences
- Creation and organisation (2013–2021) of the seminar of MAGMAT team and then ALMOST team
- Creation and organisation of the PhD students days of DAVID laboratory
- Organisation of several seminars on the interaction of social sciences and computer science for the federation "Sciences Informatiques, Humaines et Sociales (SIHS)" of the CNRS
- Creation and organisation of the seminar of the PhD students of the logic team, University Paris-Diderot
- Organisation of volontary research internships for bachelor students at University of Versailles, to develop their taste for research
- Contact for DAVID of the GDR Informatique Mathématique and GDR Théorie des Jeux (research groups of the CNRS)

Rewiews and Jurys:
- Program commitee of STACS 2022 and WEPA 2019
- Rewiewer for the journals: Annals of pure and applied logic, Computational complexity, Journal Issue of CSL, DAM, IJFCS, IPL, International journal of quantum chemistry, JAIR, LMCS, Journal Issue of MFCS, TCS, TOCS, Journal issue of WEPA
- Rewiewer for the conferences: CN, CSL, ESA, ICALP, IPEC, ISAAC, ISSAC, FOCS, FOSSACS, FSTTCS, LATA, LICS, MFCS, SAT, SODA, STACS, WG
- Member of the hiring committe for an associate professor at DAVID laboratory (2020)
- Member of the supervision committee of the PhD of Mattia Gastaldello (Lyon 1 and Rome) and Yishu Wang (Lyon 1)
- Member of the jury of the PhD of Nicolas de Rugy (Paris 7, 2015) and Yishu Wang (Lyon 1, 2021)

PhD Supervision
- Xavier Badin de Montjoye (2019-2022).
  Title: New strategy iteration methods for simple stochastic games.
  Supervisors: David Auger, Jean Michel Fourneau, Yann Strozecki. PhD co-advised at 50%.
  One article published at MFCS 2021.
- Maël Guiraud (2018-2021), PhD defended in June 2021.
  Title: Deterministic scheduling of periodic datagrams for low latency in 5G and beyond
  Academic Supervisors: Dominique Barth, Yann Strozecki.
  Industrial Supervisors: Brice Leclerc, Olivier Marcé.
  CIFRE PhD, industrial collaboration with Nokia Bell-Labs. PhD co-advised at 80% for the academic side. Two articles haves been published in network conferences, two articles have been submitted and two patents have been filed. The subject of the thesis has been presented in the

radio show "La recherche montre en main" on France Culture. Maël Guiraud has been hired as associate professor at CESI (engineering school).

Student Supervision:
- Vincent Reinhard (2012-2013, Labex CHARMMMAT), post-doctoral student on the enumeration of planar maps for cheminformatics
- Maël Guiraud supervised as a research engineer on three projects: time parallel simulation of Markov processes (2016, ANR MARMOT), latency in optic networks (2017-2018, ANR NGREEN), and optimisation of a logistic chain (2021, ANR HYPHES)
- Priscille Daoulas (2021), master internship on algorithms for building molecular cages with a given shape
- Xavier Badin de Montjoye (2018), master internship on strategy iteration algorithms for simple stochastic games
- Maël Guiraud (2016), master internship on the minimisation of latency for periodic communications in networks
- Pierre Macherel (2015), master internship on simple stochastic games
- Maxime Trekka (DIGICOSME, 2017), master 1 internship on reconstruction of individual routes in a road network from flows
- Yannis Juglaret (2014), engineering shool internship, on simple stochastic games
- Ruben Staub (2016), bachelor internship (ENS Lyon), on efficient algorithms for planar map isomorphism
- Karim Ghallab (2016), IUT internship, development of a simulator of an optic ring network to evaluate latency
- Supervised many master 1 projects for a course on topics like clustering for genomics, enumeration, scheduling, data compression, combinatorics . . .

International:
- Invited at the Dagsthul seminars "Counting Complexity 2", "Algorithmic Enumeration: Output-sensitive, Input-Sensitive, Parameterized, Approximative" (keynote speaker) and "Enumeration in Data Management" (tutorial speaker)
- Invited one month at IMSC Chennai to work with Meena Mahajan on enumeration and algebraic complexity
- Invited 1 month at the Fields Institute of Toronto (program on contraint satisfaction problems and universal algebra)
- Invited 6 months in the Theory Group of Toronto University to work with Pascal Koiran on algebraic complexity
- Two months of master 1 internship with Peter Bürgisser, Paderborn University, on Algebraic complexity and fast matrix multiplication.

Scientific Projects and Grants:
- ANR project laboratoire commun Hyphes (2021-2025), industrial collaboration between DAVID and the start-up DCbrain on algorithmic and game theory to solve coupled networks problems (telecommunication, energy, transportation . . .). Member of the board of the common laboratory. Writing of the scientific project. Organisation of the interactions between DCbrain and DAVID.
- Member of ANR project AGGREG (2014-2019) on aggregating query answers, responsible for the theme complexity and lower bounds. Participation to all the project meetings and workshops as speaker. Mid-project presentation in front of the ANR committee.
- Member of ANR projects COMPA (2014-2019) on algebraic complexity, ENUM on enumeration,

and VERAP on approximate verification. Several papers produced for these projects, participation to some their meetings and reading groups.
- Member of a PRMO project on stochastic optimisation (2017-2018).
- Member of a CEFIPRA project (2012-2015) for collaboration of indian and french researchers on algebraic complexity (writing parts of the scientific project).

Teaching Experience

I have taught 2600 hours of lectures and TDs at Versailles University, Paris-Sud University, Polytech Paris-Sud and Paris Diderot university. I have taught the following topics and created several of the courses (programming, algorithms, decidability and algorithmic game theory):

Master: Algorithmic game theory. Algorithms. Operational research, optimisation and randomized algorithms. Databases. XML and web programming. Compilers. Simulation. Supervised research projects.

Bachelor: Algorithms and data structures. Graph algorithms. Graph theory. Advanced algorithms. Language theory. Compilers. Decidability and complexity. Python programming. C programming. Object oriented programming (JAVA). Rust programming. System and logic. Introduction to computer science. Data representation. Supervised research projects.

Misc.: Formation of high school teachers (automata, complexity, decidability, advanced algorithms) to enable them to teach computer science (optional courses for lycée and classes préparatoires). Khôlles of mathematics (oral examination) in MPSI.

Teaching responsabilities:
- In charge of first year courses with 200 to 600 students.
- In charge of students with disabilities for the bachelor.
- President of Jury for first year students.
- Tutor for first year students.
- Tutor for a PhD student.

List of publications

Find here a list of my publications, with the CORE or ERA ranking when available as requested by the university. All my articles, published or submitted can be found on my webpage https://yann-strozecki.github.io/textesmaths.html. On the same page, you can also find source codes for all my experimental works.

Journal Articles

[CS21a]   Florent Capelli and Yann Strozecki. "Enumerating models of DNF faster: Breaking the dependency on the formula size". In: *Discrete Applied Mathematics (ERA Ranking A)* 303 (2021), pp. 203–215.

[Cha+21]   Arkadev Chattopadhyay, Bruno Grenet, Pascal Koiran, Natacha Portier, and Yann Strozecki. "Computing the multilinear factors of lacunary polynomials without heights". In: *Journal of Symbolic Computation (ERA Ranking A)* 104 (2021), pp. 183–206.

[CS19]   Florent Capelli and Yann Strozecki. "Incremental delay enumeration: Space and time". In: *Discrete Applied Mathematics (ERA Ranking A)* 268 (2019), pp. 179–190.

[SM19]   Yann Strozecki and Arnaud Mary. "Efficient enumeration of solutions produced by closure operations". In: *Discrete Mathematics & Theoretical Computer Science (CORE Ranking B)* 21.3 (2019).

[Str13b]   Yann Strozecki. "On Enumerating Monomials and Other Combinatorial Structures by Polynomial Interpolation". In: *Theory of Computing Systems (CORE Ranking C)* 53.4 (2013), pp. 532–568.

[SS12]   Joseph Salmon and Yann Strozecki. "Patch reprojections for Non-Local methods". In: *Signal Processing (ERA Ranking A)* 92.2 (2012), pp. 477–489.

[Str11]   Yann Strozecki. "Monadic second-order model-checking on decomposable matroids". In: *Discrete Applied Mathematics (CORE Ranking A)* 159.10 (2011), pp. 1022–1039.

Survey Articles

[Str19]   Yann Strozecki. "Enumeration Complexity". In: *Bulletin of the European Association for Theoretical Computer Science* 1.129 (2019).

Conference Proceedings

[ABS21]   David Auger, Xavier Badin de Montjoye, and Yann Strozecki. "A Generic Strategy Improvement Method for Simple Stochastic Games". In: *International Symposium on Mathematical Foundations of Computer Science, MFCS 2021 (CORE Ranking A)*. 2021.

[ACS19]   David Auger, Pierre Coucheney, and Yann Strozecki. "Solving Simple Stochastic Games with Few Random Nodes Faster Using Bland's Rule". In: *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019 (CORE Ranking A)*. 2019, 9:1–9:16.

[BGS19]   Dominique Barth, Maël Guiraud, and Yann Strozecki. "Deterministic Contention Management for Low Latency Cloud RAN over an Optical Ring". In: *Optical Network Design and Modeling - 23rd IFIP WG 6.10 International Conference, ONDM 2019*. Vol. 11616. Lecture Notes in Computer Science. Springer, 2019, pp. 479–491.

[Bar+18]   Dominique Barth, Maël Guiraud, Brice Leclerc, Olivier Marcé, and Yann Strozecki. "Deterministic scheduling of periodic messages for cloud RAN". In: *2018 25th International Conference on Telecommunications, ICT 2018 (ERA Ranking C)*. IEEE. 2018, pp. 405–410.

[MS16a]   Arnaud Mary and Yann Strozecki. "Efficient Enumeration of Solutions Produced by Closure Operations". In: *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016 (CORE Ranking A)*. 2016.

[Bar+15]   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, SEA 2015 (CORE Ranking B)*. Springer International Publishing. 2015, pp. 235–246.

[ACS14]   David Auger, Pierre Coucheney, and Yann Strozecki. "Finding Optimal Strategies of Almost Acyclic Simple Stochastic Games". In: *International Conference on Theory and Applications of Models of Computation, TAMC 2014 (CORE Ranking B)*. Springer. 2014, pp. 67–85.

[Cha+13]   Arkadev Chattopadhyay, Bruno Grenet, Pascal Koiran, Natacha Portier, and Yann Strozecki. "Factoring bivariate lacunary polynomials without heights". In: *Proceedings of the 38th international symposium on International symposium on symbolic and algebraic computation, ISSAC 2013 (CORE Ranking A)*. ACM. 2013, pp. 141–148.

[PDS12]   Sylvain Peyronnet, Michel De Rougemont, and Yann Strozecki. "Approximate verification and enumeration problems". In: *International Colloquium on Theoretical Aspects of Computing, ICTAC 2012 (CORE Ranking B)*. Springer. 2012, pp. 228–242.

[DS11a]   Arnaud Durand and Yann Strozecki. "Enumeration Complexity of Logical Query Problems with Second-order Variables". In: *EACSL Annual Conference on Computer Science Logic, CSL 2011 (CORE Ranking B)*. 2011, pp. 189–202.

[DS11b]   David Duris and Yann Strozecki. "The complexity of acyclic subhypergraph problems". In: *International Workshop on Algorithms and Computation, WALCOM 2011 (CORE Ranking B)*. Springer. 2011, pp. 45–56.

[Gre+11]   Bruno Grenet, Pascal Koiran, Natacha Portier, and Yann Strozecki. "The Limited Power of Powering: Polynomial Identity Testing and a Depth-four Lower Bound for the Permanent". In: *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011 (CORE Ranking B)*. Vol. 13. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. 2011, p. 16.

[SS10]   Joseph Salmon and Yann Strozecki. "From patches to pixels in non-local methods: Weighted-average reprojection". In: *2010 IEEE International Conference on Image Processing, ICIP 2010 (CORE Ranking B)*. IEEE. 2010, pp. 1929–1932.

[Str10b]   Yann Strozecki. "Enumeration of the monomials of a polynomial and related complexity classes". In: *International Symposium on Mathematical Foundations of Computer Science, MFCS 2010 (CORE Ranking A)*. Springer. 2010, pp. 629–640.

Conference without Proceedings

[MS16b]   Arnaud Mary and Yann Strozecki. *Generating maximal solutions given by closure operations*. Workshop on Enumeration Problems and Applications. 2016.

[Str10c] Yann Strozecki. *Monadic second-order model-checking on decomposable matroids*. Satellite Workshop of CSL/MFCS: Logic, Combinatorics and Computation. 2010.

Submitted Articles and Manuscripts

[Bar+21] Dominique Barth, Maël Guiraud, Brice Leclerc, Olivier Marcé, and Yann Strozecki. "Scheduling Synchronized Periodic Messages on a Meshed Network". 2021.

[BGS21]   Dominique Barth, Maël Guiraud, and Yann Strozecki. "Deterministic Scheduling of Periodic Messages for Cloud RAN". 2021. URL: https://arxiv.org/abs/1801.07029.

[CS21b]   Florent Capelli and Yann Strozecki. "Geometric Amortization of Enumeration Algorithms". 2021. URL: https://arxiv.org/abs/2108.10208.

[GS21]   Maël Guiraud and Yann Strozecki. "Scheduling Periodic Messages on a Shared Link". 2021. URL: https://arxiv.org/abs/2002.07606.

[Bar+20] Dominique Barth, Olivier R.P. David, Franck Quessette, Yann Strozecki, and Sandrine Vial. "In Silico Discovery of Complex Molecular Containers". 2020. URL: https://yann-strozecki.github.io/Insilico.pdf.

[Str13a] Yann Strozecki. "Decomposition-width: Extending Clique-width to Hypergraphs". 2013. URL: https://yann-strozecki.github.io/hypergraph_decomposition.pdf.

[Str10a] Yann Strozecki. "Enumeration complexity and matroid decomposition". PhD thesis. Université Paris Diderot - Paris 7, 2010.

[Str07]   Yann Strozecki. "Algorithmes holographiques". MA thesis. Université Paris-Diderot, 2007.