Paradigms for Parameterized Enumeration*

Nadia Creignou[1], Arne Meier[2], Julian-Steffen Müller[2], Johannes Schmidt[3], and Heribert Vollmer[2]

[1] Aix-Marseille Université
nadia.creignou@lif.univ-mrs.fr
[2] Leibniz Universität Hannover
{meier,mueller,vollmer}@thi.uni-hannover.de
[3] Linköping University
johannes.schmidt@liu.se

Abstract. The aim of the paper is to examine the computational complexity and algorithmics of enumeration, the task to output all solutions of a given problem, from the point of view of parameterized complexity. First we define formally different notions of efficient enumeration in the context of parameterized complexity. Second we show how different algorithmic paradigms can be used in order to get parameter-efficient enumeration algorithms in a number of examples. These paradigms use well-known principles from the design of parameterized decision as well as enumeration techniques, like for instance kernelization and self-reducibility. The concept of kernelization, in particular, leads to a characterization of fixed-parameter tractable enumeration problems.

1 Introduction

This paper is concerned with algorithms for and complexity studies of enumeration problems, the task of generating all solutions of a given computational problem. The area of enumeration algorithms has experienced tremendous growth over the last decade. Prime applications are query answering in databases and web search engines, data mining, web mining, bioinformatics and computational linguistics.

Parameterized complexity theory provides a framework for a refined analysis of hard algorithmic problems. It measures complexity not only in terms of the input size, but in addition in terms of a parameter. Problem instances that exhibit structural similarities will have the same or similar parameter(s). Efficiency now means that for fixed parameter, the problem is solvable with reasonable time resources. A parameterized problem is fixed-parameter tractable (in FPT) if it can be solved in polynomial time for each fixed value of the parameter, where the degree of the polynomial does not depend on the parameter. Much like in the classical setting, to give evidence that certain algorithmic problems are not in

* Supported by a Campus France/DAAD Procope grant, Campus France Projet No 28292TE, DAAD Projekt-ID 55892324.

FPT one shows that they are complete for superclasses of FPT, like the classes in what is known as the W-hierarchy.

Our main goal is to initiate a study of enumeration from a parameterized complexity point of view and in particular to develop parameter-efficient enumeration algorithms. Preliminary steps in this direction have been undertaken by H. Fernau [5]. He considers algorithms that output *all* solutions of a problem to a given instance in polynomial time for each fixed value of the parameter, where, as above, the degree of the polynomial does not depend on the parameter (let us briefly call this fpt-time). We subsume problems that exhibit such an algorithm in the class Total-FPT. (A similar notion was studied by Damaschke [4]). Algorithms like these can of course only exists for algorithmic problems that possess only relatively few solutions for an input instance. We therefore consider algorithms that exhibit a delay between the output of two different solutions of fpt-time, and we argue that this is the "right way" to define tractable parameterized enumeration. The corresponding complexity class is called Delay-FPT.

We then study the techniques of kernelization (stemming from parameterized complexity) and self-reducibility (well-known in the design of enumeration algorithms) under the question if they can be used to obtain parameter-efficient enumeration algorithms. We study these techniques in the context of different algorithmic problems from the context of propositional satisfiability (and vertex cover, which can, of course, also be seen as a form of weighted 2-CNF satisfiability question). We obtain a number of upper and lower bounds on the enumerability of these problems.

In the next section we introduce parameterized enumeration problems and suggest four hopefully reasonable complexity classes for their study. In the following two sections we study in turn kernelization and self-reducibility, and apply them to the problems Vertex-Cover, MaxOnes-SAT and detection of strong Horn-backdoor sets. We conclude with some open questions about related algorithmic problems.

2 Complexity Classes for Parameterized Enumeration

Because of the amount of solutions that enumeration algorithms possibly produce, the size of their output is often much larger (e.g., exponentially larger) than the size of their input. Therefore, polynomial time complexity is not a suitable yardstick of efficiency when analyzing their performance. As it is now agreed, one is more interested in the regularity of these algorithms rather than in their total running time. For this reason, the efficiency of an enumeration algorithm is better measured by the delay between two successive outputs, see e.g., [7]. The same observation holds within the context of parametrized complexity and we can define parameterized complexity classes for enumeration based on this time elapsed between two successive outputs. Let us start with the formal definition of a parameterized enumeration problem.

Definition 1. *A* parameterized enumeration problem *(over a finite alphabet Σ) is a triple* $E = (Q, \kappa, \mathrm{Sol})$ *such that*

- $Q \subseteq \Sigma^*$,
- *κ is a parameterization of Σ^*, that is $\kappa\colon \Sigma^* \to \mathbb{N}$ is a polynomial time computable function.*
- $\mathrm{Sol} : \Sigma^* \to \mathcal{P}(\Sigma^*)$ *is a function such that for all $x \in \Sigma^*$, $\mathrm{Sol}(x)$ is a finite set and $\mathrm{Sol}(x) \neq \emptyset$ if and only if $x \in Q$.*

If $E = (Q, \kappa, \mathrm{Sol})$ is a parameterized enumeration problem over the alphabet Σ, then we call strings $x \in \Sigma^*$ instances of E, the number $\kappa(x)$ the corresponding parameter, and $\mathrm{Sol}(x)$ the set of solutions of x. As an example we consider the problem of enumerating all vertex covers with bounded size of a graph.

Problem:	All-Vertex-Cover
Input:	An undirected graph G and a positive integer k
Parameter:	k
Output:	The set of all vertex covers of G of size $\leq k$

An *enumeration algorithm* $\mathcal{A}$ for the enumeration problem $E = (Q, \kappa, \mathrm{Sol})$ is an algorithm, which on the input x of E, outputs exactly the elements of $\mathrm{Sol}(x)$ without duplicates, and which terminates after a finite number of steps on every input.

At first we need to fix the notion of delay for algorithms.

Definition 2 (Delay). *Let $E = (Q, \kappa, \mathrm{Sol})$ be a parameterized enumeration problem and $\mathcal{A}$ an enumeration algorithm for E. Let $x \in Q$, then we say that the i-th delay of $\mathcal{A}$ is the time between outputting the i-th and $(i+1)$-st solutions in* $\mathrm{Sol}(x)$. *Further, we define the 0-th delay as the* precalculation time *as the time from the start of the computation to the first output statement. Analogously, the n-th delay, for $n = |\mathrm{Sol}(x)|$, is the* postcalculation time *which is the time needed after the last output statement until $\mathcal{A}$ terminates.*

We are now ready to define different notions of fixed-parameter tractability for enumeration problems.

Definition 3. *Let $E = (Q, \kappa, \mathrm{Sol})$ be a parameterized enumeration problem and $\mathcal{A}$ an enumeration algorithm for E.*

1. *The algorithm $\mathcal{A}$ is a* Total-FPT *algorithm if there exist a computable function $t\colon \mathbb{N} \to \mathbb{N}$ and a polynomial p such that for every instance $x \in \Sigma^*$, $\mathcal{A}$ outputs all solutions of* $\mathrm{Sol}(x)$ *in time at most $t(\kappa(x)) \cdot p(|x|)$.*
2. *The algorithm $\mathcal{A}$ is a* Delay-FPT *algorithm if there exist a computable function $t\colon \mathbb{N} \to \mathbb{N}$ and a polynomial p such that for every $x \in \Sigma^*$, $\mathcal{A}$ outputs all solutions of* $\mathrm{Sol}(x)$ *with delay of at most $t(\kappa(x)) \cdot p(|x|)$.*

Though this will not be in the focus of the present paper, we remark that, in analogy to the non-parameterized case (see [3,15]), one can easily adopt the definition for Inc-FPT algorithms whose ith delay is at most $t(\kappa(x)) \cdot p(|x| + i)$. Similarly, one gets the notion of Output-FPT algorithms which is defined by a runtime of at most $t(\kappa(x)) \cdot p(|x| + |\mathrm{Sol}(x)|)$.

Definition 4. *The class* Total-FPT *(resp.,* Delay-FPT*) is the class of all parameterized enumeration problems that admit a* Total-FPT *(resp.,* Delay-FPT*) enumeration algorithm.*

Observe that Fernau's notion of fixed parameter enumerable [5] is equivalent to our term of Total-FPT. Obviously the existence of a Total-FPT enumeration algorithm requires that for every instance x the number of solution is bounded by $f(\kappa(x)) \cdot p(|x|)$, which is quite restrictive. Nevertheless, Fernau was able to show that the problem Minimum-Vertex-Cover (where we are only interested in vertex covers of minimum cardinality) is in Total-FPT, but by the just given cardinality constraint, All-Vertex-Cover is not in Total-FPT. In the upcoming section we will prove that All-Vertex-Cover is in Delay-FPT; hence we conclude:

Corollary 5. Total-FPT $\subsetneq$ Delay-FPT.

We consider that Delay-FPT should be regarded as the good notion of tractability for parameterized enumeration complexity.

3 Enumeration by Kernelization

Kernelization is one of the most successful techniques in order to design parameter-efficient algorithms, and actually characterizes parameter-tractable problems. Remember that kernelization consists in a pre-processing, which is a polynomial time many-one reduction of a problem to itself with the additional property that the (size of the) image is bounded in terms of the parameter of the argument (see e.g., [6]).

In the following we propose a definition of an *enum-kernelization*, which should be seen as a pre-processing step suitable for an efficient enumeration.

Definition 6. *Let $(Q, \kappa, \mathrm{Sol})$ be a parameterized enumeration problem over Σ. A polynomial time computable function $K\colon \Sigma^* \to \Sigma^*$ is an* enum-kernelization *of $(Q, \kappa, \mathrm{Sol})$ if there exist:*

1. *a computable function $h\colon \mathbb{N} \to \mathbb{N}$ such that for all $x \in \Sigma^*$ we have*
$$(x \in Q \Leftrightarrow K(x) \in Q) \text{ and } |K(x)| \leq h(\kappa(x)),$$
2. *a computable function $f\colon {\Sigma^*}^2 \to \mathcal{P}(\Sigma^*)$, which from a pair (x, w) where $x \in Q$ and $w \in \mathrm{Sol}(K(x))$, computes a subset of $\mathrm{Sol}(x)$, such that*
   (a) for all $w_1, w_2 \in \mathrm{Sol}(K(x))$, $w_1 \neq w_2 \Rightarrow f(x, w_1) \cap f(x, w_2) = \emptyset$,
   (b) $$\bigcup_{w \in \mathrm{Sol}(K(x))} f(x, w) = \mathrm{Sol}(x)$$
   (c) there exists an enumeration algorithm $\mathcal{A}_f$, which on input (x, w), where $x \in Q$ and $w \in \mathrm{Sol}(K(x))$, enumerates all solutions of $f(x, w)$ with delay $p(|x|) \cdot t(\kappa(x))$, where p is a polynomial and t is a computable function.

If K is an enum-kernelization of $(Q, \kappa, \mathrm{Sol})$, then for every instance x of Q the image $K(x)$ is called an enum-kernel *of x (under K).*

An enum-kernelization is a reduction K from a parameterized enumeration problem to itself. As in the decision setting it has the property that the image is bounded in terms of the parameter argument. For a problem instance x, $K(x)$ is the kernel of x. Observe that if K is an enum-kernelization of the enumeration problem $(Q, \kappa, \mathrm{Sol})$, then it is also a kernelization for the associated decision problem. In order to fit for enumeration problems, enum-kernelizations have the additional property that the set of solutions of the original instance x can be rebuilt from the set of solutions of the image $K(x)$ with Delay-FPT. This can be seen as a generalization of the notion of *full kernel* from [4], appearing in the context of what is called *subset minimization problems*. A full kernel is a kernel that contains all minimal solutions, since they represent in a certain way all solutions. In the context of backdoor sets (see the next section), what is known as a loss-free kernel [13] is a similar notion. In our definition, an enum-kernel is a kernel that represents all solutions in the sense that they can be obtained with FPT delay from the solutions for the kernel.

Vertex cover is a very famous problem whose parameterized complexity has been extensively studied. It is a standard example when it comes to kernelization. Let us examine it in the light of the notion of enum-kernelization.

Proposition 7. All-Vertex-Cover *has an enum-kernelization.*

Proof. Given a graph $G = (V, E)$ and a positive integer k, we are interested in enumerating all vertex covers of G of size at most k. We prove that the famous Buss' kernelization [6, pp. 208ff] provides an enum-kernelization. Let us remember that Buss' algorithm consists in applying repeatedly the following rules until no more reduction can be made:

1. If v is a vertex of degree greater than k, remove v from the graph and decrease k by one.
2. If v is an isolated vertex, remove it.

The algorithm terminates and the kernel $K(G)$ is the reduced graph (V_K, E_K) so obtained if it has less than k^2 edges, and the complete graph $\mathcal{K}_{k+1}$ otherwise.

One verifies that whenever in a certain step of the removing process rule (1) is applicable to a vertex v, and v is not removed immediately, then rule (1) remains applicable to v also in any further step, until it is removed. Therefore, whenever we have a choice during the removal process, our choice does not influence the finally obtained graph: the kernel is unique.

Suppose that $K(G) = (V_K, E_K)$. Let V_D be the set of vertices (of large degree) that are removed by the rule (1) and V_I the set of vertices (isolated) that are removed by the rule (2). On the one hand every vertex cover of size $\leq k$ of G has to contain V_D. On the other hand, no vertex from V_I is part of a minimal vertex cover. Thus, all vertex covers of G are obtained in considering all the vertex covers of $K(G)$, completing them by V_D and by some vertices of V_I up to the cardinality k. Therefore, given W a vertex cover of $K(G)$, then we define $f(G, W) = \{W \cup V_D \cup V' \mid V' \subseteq V_I, |V'| \leq k - |W| - |V_D|\}$. It is then clear that for $W_1 \neq W_2$, $W_1, W_2 \in \mathrm{Sol}(K(G))$, we have that $f(G, W_1) \cap f(G, W_2) = \emptyset$. From the discussion above we have that $\bigcup_{W\in\mathrm{Sol}(K(G))} f(G,W)$ is the set of all $\leq k$-vertex covers of G. Finally, given W a vertex cover of $K(G)$, after a polynomial time pre-processing of G by Buss's kernelization in order to compute V_D and V_I, the enumeration of $f(G,W)$ comes down to an enumeration of all subsets of V_I of size at most $k-|W|-|V_D|$. Such an enumeration can be done with polynomial delay by standard algorithms. Therefore, the set $f(G,W)$ can be enumerated with polynomial delay and, *a fortiori*, with Delay-FPT. □

As in the context of decision problems, enum-kernelization actually characterizes the class of enumeration problems having Delay-FPT-algorithm, as shown in the following theorem.

Theorem 8. *For every parameterized enumeration problem* (Q,κ,Sol) *over* Σ*, the following are equivalent:*

1. (Q,κ,Sol) *is in* Delay-FPT
2. *For all* $x\in\Sigma^*$ *the set* $\mathrm{Sol}(x)$ *is computable and* (Q,κ,Sol) *has an enum-kernelization.*

Proof. (2) $\Rightarrow$ (1)**:** Let K be an enum-kernelization of (Q,κ,Sol). Given an instance $x\in\Sigma^*$ the following algorithm enumerates all solution in $\mathrm{Sol}(x)$ with Delay-FPT: compute $K(x)$ in polynomial time, say $p'(|x|)$. Compute $\mathrm{Sol}(K(x))$, this requires a time $g(\kappa(x))$ for some function g since the size of $K(x)$ is bounded in terms of the parameter argument. Apply successively the enumeration algorithm $\mathcal{A}_f$ to the input (x,w) for each $w\in\mathrm{Sol}(K(x))$. Since $\mathcal{A}_f$ requires a delay $p(|x|)\cdot t(\kappa(x))$, the delay of this enumeration algorithm is bounded from above by $(p'(|x|)+p(|x|))\cdot(g(\kappa(x))+t(\kappa(x)))$. The correctness of the algorithm follows from the definition of an enum-kernelization (Item 2.(a) ensures that there is no repetition, Item 2.(b) that all solutions are output).

(1) $\Rightarrow$ (2)**:** Let $\mathcal{A}$ be an enumeration algorithm for (Q,κ,Sol) that requires delay $p(n)\cdot t(k)$ where p is a polynomial and t some computable function. Without loss of generality we assume that $p(n)\geq n$ for all positive integer n. If $Q=\emptyset$ or $Q=\Sigma^*$ then (Q,κ,Sol) has a trivial kernelization that maps every $x\in\Sigma^*$ to the empty string ϵ. If $Q=\emptyset$ we are done. If $Q=\Sigma^*$, then fix $w_\epsilon\in\mathrm{Sol}(\epsilon)$ and set for all x, $f(x,w_\epsilon)=\mathrm{Sol}(x)$ and $f(x,w)=\emptyset$ for $w\in\mathrm{Sol}(\epsilon)\setminus\{w_\epsilon\}$. Otherwise, we fix $x_0\in\Sigma^*\setminus Q$, and $x_1\in Q$ with $w_1\in\mathrm{Sol}(x_1)$.

The following algorithm $\mathcal{A}'$ computes an enum-kernelization for (Q,κ,Sol): Given $x\in\Sigma^*$ with $n:=|x|$ and $k=\kappa(x)$,

1. the algorithm simulates $p(n)\cdot p(n)$ steps of $\mathcal{A}$.
2. If it stops with the answer "no solution", then set $K(x)=x_0$ (since $x_0\notin Q$, the function f does not need to be defined).
3. If a solution is output within this time, then set $K(x)=x_1$, $f(x,w_1)=\mathrm{Sol}(x)$ and $f(x,w)=\emptyset$ for all $w\in\mathrm{Sol}(x_1)\setminus\{w_1\}$.
4. If it does not output a solution within this time, then it holds $n\leq p(n)\leq t(k)$ and then we set $K(x)=x$, and $f(x,w)=\{w\}$ for all $w\in\mathrm{Sol}(x)$.

Clearly $K(x)$ can thus be computed in time $p(n)^2$, $|K(x)| \leq |x_0|+|x_1|+t(k)$, ($x \in Q \Leftrightarrow K(x) \in Q$), and the function f we have obtained satisfies all the requirements of Theorem 6, in particular the enumeration algorithm $\mathcal{A}$ can be used to enumerate $f(x,w)$ when applicable. Therefore K provides indeed an enum-kernelization for $(Q, \kappa, \mathrm{Sol})$. □

Corollary 9. All-Vertex-Cover *is in* Delay-FPT.

Remark 10. Observe that in the proof of Theorem 7, the enumeration of the sets of solutions obtained from a solution W of $K(G)$ is enumerable even with polynomial-delay, we do not need fpt delay. We will show in the full paper that this is a general property: Enum-kernelization can be equivalently defined as FPT-preprocessing followed by enumeration with polynomial delay.

4 Enumeration by Self-reducibility

In this section we would like to exemplify the use of the algorithmic paradigm of self-reducibility ([16,8,15]), on which various enumeration algorithms are based in the literature. The self-reducibility property of a problem allows a "search-reduces-to-decision" algorithm to enumerate the solutions. This technique seems quite appropriate for satisfiability related problems. We will first investigate the enumeration of models of a formula having weight at least k, and then turn to strong HORN-backdoor sets of size k. In the first example the underlying decision problem can be solved in using kernelization (see [9]), while in the second it is solved in using the bounded-search-tree technique.

4.1 Enumeration Classification for MaxOnes-SAT

The self-reducibility technique was in particular applied in order to enumerate all satisfying assignments of a generalized CNF-formula [1], thus allowing to identify classes of formulas which admit efficient enumeration algorithms. In the context of parameterized complexity a natural problem is MaxOnes-SAT, in which the question is to decide whether there exists a satisfying assignment of weight at least k, the integer k being the parameter. We are here interested in the corresponding enumeration problem, and we will study it for generalized CNF formulas, namely in Schaefer's framework. In order to state the problem we are interested in more formally, we need some notation.

A *logical relation* of arity k is a relation $R \subseteq \{0,1\}^k$. By abuse of notation we do not make a difference between a relation and its predicate symbol. A *constraint*, C, is a formula $C = R(x_1, \ldots, x_k)$, where R is a logical relation of arity k and the x_i's are (not necessarily distinct) variables. If u and v are two variables, then $C[u/v]$ denotes the constraint obtained from C in replacing each occurrence of v by u. An assignment m of truth values to the variables *satisfies* the constraint C if $\big(m(x_1), \ldots, m(x_k)\big) \in R$. A *constraint language* Γ is a finite set of logical relations. A *Γ-formula* ϕ, is a conjunction of constraints using only logical relations from Γ and is hence a quantifier-free first order formula.

With $\mathrm{Var}(\phi)$ we denote the set of variables appearing in ϕ. A Γ-formula ϕ is satisfied by an assignment $m : \mathrm{Var}(\phi) \to \{0, 1\}$ if m satisfies all constraints in ϕ simultaneously (such a satisfying assignment is also called a *model* of ϕ). The *weight of a model* is given by the number of variables set to true. Assuming a canonical order on the variables we can regard models as tuples in the obvious way and we do not distinguish between a formula ϕ and the logical relation R_ϕ it defines, i.e., the relation consisting of all models of ϕ. In the following we will consider two particular constraints, namely $\mathrm{Imp}(x, y) = (x \to y)$ and $\mathrm{T}(x) = (x)$.

We are interested in the following parameterized enumeration problem.

Problem:	Enum-MaxOnes-SAT(Γ)
Input:	A Γ-formula φ and a positive integer k
Parameter:	k
Output:	All assignments satisfying φ of weight $\geq k$

The corresponding decision problem, denoted by MaxOnes-SAT(Γ), i.e., the problem to decide if a given formula has a satisfying assignment of a given weight, has been studied by Kratsch et al. [9]. They completely settle the question of its parameterized complexity in Schaefer's framework. To state their result we need some terminology concerning types of Boolean relations.

Well known already from Schaefer's original paper [14] are the following seven classes: We say that a Boolean relation R is *a-valid* (for $a \in \{0, 1\}$) if $R(a, \ldots, a) = 1$. A relation R is *Horn* (resp., *dual Horn*) if R can be defined by a CNF formula which is Horn (resp., dual Horn), i.e., every clause contains at most one positive (resp., negative) literal. A relation R is *bijunctive* if R can be defined by a 2-CNF formula. A relation R is *affine* if it can be defined by an *affine* formula, i.e., conjunctions of XOR-clauses (consisting of an XOR of some variables plus maybe the constant 1)—such a formula may also be seen as a system of linear equations over GF[2]. A relation R is *complementive* if for all $m \in R$ we have also $\mathbf{1} \oplus m \in R$.

Kratsch et al. [9] introduce a new restriction of the class of bijunctive relations as follows. For this they use the notion of *frozen implementation*, stemming from [12]. Let φ be a formula and $x \in \mathrm{Var}(\varphi)$, then x is said to be *frozen* in φ if it is assigned the same truth value in all its models. Further, we say that Γ *freezingly implements* a given relation R if there is a Γ-formula φ such that $R(x_1, \ldots x_n) \equiv \exists X \varphi$, where φ uses variables from $X \cup \{x_1, \ldots x_n\}$ only, and all variables in X are frozen in φ. For sake of readability, we denote by $\langle \Gamma \rangle_{fr}$ the set of all relations that can be freezingly implemented by Γ. A relation R is *strongly bijunctive* if it is in $\langle \{(x \vee y), (x \neq y), (x \to y)\} \rangle_{fr}$.

Finally, we say that a constraint language Γ has one of the just defined properties if every relation in Γ has the property.

Theorem 11. *[9, Thm. 7] If Γ is 1-valid, dual-Horn, affine, or strongly bijunctive, then* MaxOnes-SAT(Γ) *is in* FPT. *Otherwise* MaxOnes-SAT(Γ) *is* W[1]*-hard.*

Interestingly we can get a complete classification for enumeration as well. The fixed-parameter efficient enumeration algorithms are obtained through the algorithmic paradigm of self-reducibility.

We would like to mention that an analogously defined decision problem MinOnes-SAT(Γ) is in FPT (by a bounded search-tree algorithm) and the enumeration problem has FPT-delay for all constraint langauges Γ. The decision problem ExactOnes-SAT(Γ) has been studied by Marx [10] and shown to be in FPT iff Γ has a property called "weakly separable". We remark that it can be shown, again by making use of self-reducibility, that under the same conditions, the corresponding enumeration algorithm has FPT-delay. This will be presented in the full paper. In the present submission we concentrate on the, as we think, more interesting maximization problem, since here, the classification of the complexity of the enumeration problem differs from the one for the decision problem, as we state in the following theorem.

Theorem 12. *If Γ is dual-Horn, affine, or strongly bijunctive, then there is a* Delay-FPT *algorithm for* Enum-MaxOnes-SAT(Γ). *Otherwise such an algorithm does not exist unless* W[1] = FPT.

It would be interesting for those cases of Γ that do not admit a Delay-FPT algorithm to determine an upper bound besides the trivial exponential time bound to enumerate all solutions. In particular, are there such sets Γ for which Enum-MaxOnes-SAT(Γ) is in Output-FPT?

Proof. (of Theorem 12) We first propose a canonical algorithm for enumerating all satisfying assignments of weight at least k. The function `HasMaxOnes`(ϕ, k) tests if the formula ϕ has a model of weight at least k.

Algorithm 2. Enumerate the models of weight at least k

```
  Input: A formula $\phi$ with $\mathrm{Var}(\phi) = \{x_1, \ldots, x_n\}$, an integer $k$
  Output: All sat. assignments (given as sets of variables) of $\phi$ of weight $\geq k$.
1 if HasMaxOnes($\phi, k$) then Generate($\phi, \emptyset, k, n$)

  Procedure Generate($\phi, M, w, p$) :
1 if $w = 0$ or $p = 0$ then return $M$
2 else
3     if HasMaxOnes($\phi[x_p = 1], w - 1$) then
4         Generate($\phi[x_p = 1], M \cup \{x_p\}, w - 1, p - 1$)
5     if HasMaxOnes($\phi[x_p = 0], w$) then Generate($\phi[x_p = 0], M, w, p - 1$)
```

Observe that if Γ is dual-Horn, affine, or strongly bijunctive, then according to Theorem 12 the procedure `HasMaxOnes`(ϕ, k) can be performed in FPT. Moreover essentially if ϕ is dual-Horn (resp., affine, strongly bijunctive) then so are $\phi[x_p = 0]$ and $\phi[x_p = 1]$ for any variable x_p. Therefore, in all these cases the proposed enumeration algorithm has clearly Delay-FPT. □

A full version of the proof can be found in the arXiv [2].

4.2 Enumeration of Strong HORN-Backdoor Sets

We consider here the enumeration of strong backdoor sets. Let us introduce some relevant terminology [18]. Consider a formula ϕ, a set V of variables of ϕ, $V \subseteq \mathrm{Var}(\phi)$. For a truth assignment τ, $\phi(\tau)$ denotes the result of removing all clauses from ϕ which contain a literal x with $\tau(x) = 1$ and removing literals y with $\tau(y) = 0$ from the remaining clauses.

The set V is a *strong* HORN-*backdoor set* of ϕ if for all truth assignment $\tau\colon V \to \{0,1\}$ we have $\phi(\tau) \in \mathrm{HORN}$. Observe that equivalently V is a strong HORN-backdoor set of ϕ if $\phi|_V$ is HORN, where $\phi|_V$ denotes the formula obtained from ϕ in deleting in ϕ all occurrences of variables from V.

Now let us consider the following enumeration problem.

Problem:	EXACT-STRONG-BACKDOORSET[HORN]
Input:	A formula ϕ in CNF
Parameter:	k
Output:	The set of all strong HORN-backdoor sets of ϕ of size exactly k

From [11] we know that detection of strong HORN-backdoor sets is in FPT. In using a variant of bounded-search tree the authors use in their FPT-algorithm, together with self-reducibility we get an efficient enumeration algorithm for all strong HORN-backdoor sets of size k.

Theorem 13. EXACT-STRONG-BACKDOORSET[HORN] *is in* Delay-FPT.

Proof. The procedure `GenerateSBDS`(ϕ, B, k, V) depicted in Algorithm 1 enumerates all sets $S \subseteq V$ of size k such that $B \cup S$ is a strong HORN-backdoor set for ϕ, while the function `Exists-SBDS`(ϕ, k, V) tests if ϕ has a strong HORN-backdoor set of size exactly k made of variables from V.

The point that this algorithm is indeed in Delay-FPT relies on the fact that the function `Exists-SBDS` depicted in Algorithm 2 is in FPT. This function is an adaptation of the one proposed in [11]. There Nishimura et al. use an important fact holding for non-HORN clauses (i.e., clauses contains at least two positive literals): if p_1, p_2 are two positive literals then either one of them must belong to any strong backdoor set of the complete formula.

In their algorithm they just go through all clauses for these occurrences. However for our task, the enumeration of the backdoor sets, it is very important to take care of the ordering of variables. The reason for this is the following. Using the algorithm without changes makes it impossible to enumerate the backdoor sets because wrong sets would be considered: e.g., for some formula ϕ and variables $x_1, \ldots, x_n$ let $B = \{x_2, x_4, x_5\}$ be the only strong backdoor set. Then, during the enumeration process, one would come to the point where the sets with x_2 have been investigated (our algorithm just enumerates from the smallest variable index to the highest). When we start investigating the sets containing x_4, the procedure would then wrongly say "yes there is a backdoor set containing x_4" which is not desired in this situation because we finished considering x_2 (and only want to investigate backdoor sets that do not contain x_2).

Algorithm 3. Enumerate all strong HORN-backdoor sets of size k

```
Input: A formula $\phi$, an integer $k$
Output: All strong HORN-backdoor sets of size $k$.
1  if Exists-SBDS($\phi, k, \mathrm{Var}(\phi)$) then GenerateSBDS($\phi, \emptyset, k, \mathrm{Var}(\phi)$)

Procedure GenerateSBDS($\phi, B, k, V$) :
1  if $k = 0$ or $V = \emptyset$ then return $B$
2  else
3      if Exists-SBDS($\phi|_{B\cup\{\min(V)\}}, k-1, V \setminus \{\min(V)\}$) then
4          GenerateSBDS($\phi, B \cup \{\min(V)\}, k-1, V \setminus \{\min(V)\}$)
5      if Exists-SBDS($\phi|_B, k, V \setminus \{\min(V)\}$) then
6          GenerateSBDS($\phi, B, k, V \setminus \{\min(V)\}$)

Function Exists-SBDS($\phi, k, V$) :
1  if $k = 0$ or $V = \emptyset$ then
2      if $\phi|_V \in$ HORN then return true else return false
3  if there is a clause $C$ with two positive literals $p_1, p_2$ then
4      if exactly one of $p_1$ and $p_2$ is in $V$, say $p_1 \in V, p_2 \notin V$ then
5          if Exists-SBDS($\phi|_{\{p_1\}}, k-1, V \setminus \{p_1\}$) then return true
6      else
7          if $p_1 \in V$ and $p_2 \in V$ then
8              if Exists-SBDS($\phi|_{\{p_1\}}, k-1, V \setminus \{p_1\}$) then return true
9              if Exists-SBDS($\phi|_{\{p_2\}}, k-1, V \setminus \{p_2\}$) then return true
10     return false
11 else return true
```

Therefore the algorithm needs to consider only the variables in the set V where in each recursive call the minimum variable (i.e., the one with smallest index) is removed from the set V of considered variables. □

5 Conclusion

We made a first step to develop a computational complexity theory for parameterized enumeration problems by defining a number of, as we hope, useful complexity classes. We examined two design paradigms for parameterized algorithms from the point of view of enumeration. Thus we obtained a number of upper bounds and also some lower bounds for important algorithmic problems, mainly from the area of propositional satisfiability.

As further promising problems we consider the cluster editing problem [4] and the k-flip-SAT problem [17].

Of course it will be very interesting to examine further algorithmic paradigms for their suitability to obtain enumeration algorithms. Here, we think of the technique of bounded search trees and the use of structural graph properties like treewidth.

Acknowledgements. We are very thankful to Frédéric Olive for helpful discussions. We also acknowledge many helpful comments from the reviewers.

References

1. Creignou, N., Hébrard, J.-J.: On generating all solutions of generalized satisfiability problems. Theoretical Informatics and Applications 31(6), 499–511 (1997)
2. Creignou, N., Meier, A., Müller, J.-S., Schmidt, J., Vollmer, H.: Paradigms for parameterized enumeration. CoRR, arXiv:1306.2171 (2013)
3. Creignou, N., Olive, F., Schmidt, J.: Enumerating all solutions of a Boolean CSP by non-decreasing weight. In: Sakallah, K.A., Simon, L. (eds.) SAT 2011. LNCS, vol. 6695, pp. 120–133. Springer, Heidelberg (2011)
4. Damaschke, P.: Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. TCS 351(3), 337–350 (2006)
5. Fernau, H.: On parameterized enumeration. Computing and Combinatorics (2002)
6. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer (2006)
7. Johnson, D.S., Papadimitriou, C.H., Yannakakis, M.: On generating all maximal independent sets. IPL 27(3), 119–123 (1988)
8. Khuller, S., Vazirani, V.V.: Planar graph coloring is not self-reducible, assuming P $\neq$ NP. TCS 88(1), 183–189 (1991)
9. Kratsch, S., Marx, D., Wahlström, M.: Parameterized complexity and kernelizability of max ones and exact ones problems. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 489–500. Springer, Heidelberg (2010)
10. Marx, D.: Parameterized complexity of constraint satisfaction problems. Computational Complexity (14), 153–183 (2005)
11. Nishimura, N., Ragde, P., Szeider, S.: Detecting backdoor sets with respect to horn and binary clauses. In: Proc. SAT (2004)
12. Nordh, G., Zanuttini, B.: Frozen boolean partial co-clones. In: Proc. ISMVL, pp. 120–125 (2009)
13. Samer, M., Szeider, S.: Backdoor trees. In: Proc. AAAI, pp. 363–368. AAAI Press (2008)
14. Schaefer, T.J.: The complexity of satisfiability problems. In: Proc. STOC, pp. 216–226. ACM Press (1978)
15. Schmidt, J.: Enumeration: Algorithms and complexity. Master's thesis, Leibniz Universität Hannover (2009)
16. Schnorr, C.P.: Optimal algorithms for self-reducible problems. In: Proc. ICALP, pp. 322–337 (1976)
17. Szeider, S.: The parameterized complexity of k-flip local search for SAT and MAX SAT. Discrete Optimization 8(1), 139–145 (2011)
18. Williams, R., Gomes, C., Selman, B.: Backdoors to typical case complexity. In: Proc. IJCAI, pp. 1173–1178 (2003)