Refined Notions of Parameterized Enumeration Kernels with Applications to Matching Cut Enumeration

Petr A. Golovach
Department of Informatics, University of Bergen, Norway

Christian Komusiewicz
Fachbereich Mathematik und Informatik, Philipps-Universität Marburg, Germany

Dieter Kratsch
LGIMP, Université de Lorraine, Metz, France

Van Bang Le
Institut für Informatik, Universität Rostock, Germany

Abstract

An enumeration kernel as defined by Creignou et al. [Theory Comput. Syst. 2017] for a parameterized enumeration problem consists of an algorithm that transforms each instance into one whose size is bounded by the parameter plus a solution-lifting algorithm that efficiently enumerates all solutions from the set of the solutions of the kernel. We propose to consider two new versions of enumeration kernels by asking that the solutions of the original instance can be enumerated in polynomial time or with polynomial delay from the kernel solutions. Using the NP-hard Matching Cut problem parameterized by structural parameters such as the vertex cover number or the cyclomatic number of the input graph, we show that the new enumeration kernels present a useful notion of data reduction for enumeration problems which allows to compactly represent the set of feasible solutions.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases enumeration problems, polynomial delay, output-sensitive algorithms, kernelization, structural parameterizations, matching cuts

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.37

Related Version *Full Version*: https://arxiv.org/abs/2101.03800

Funding *Petr A. Golovach*: Supported by the Research Council of Norway via the project "MULTIVAL" (grant no. 263317).

Acknowledgements We dedicate this paper to the memory of our coauthor and friend Dieter Kratsch who recently passed away. Without Dieter, this paper would have never been written.

1 Introduction

The enumeration of all feasible solutions of a computational problem is a fundamental task in computer science. For the majority of enumeration problems, the number of feasible solutions can be exponential in the input size in the worst-case. The running time of enumeration algorithms is thus measured not only in terms of the input size n but also in terms of the output size. The two most-widely used definitions of efficient algorithms are polynomial output-sensitive algorithms where the running time is polynomial in terms of input and output size and polynomial-delay algorithms, where the algorithm spends only a polynomial running time between the output of consecutive solutions. Since in some enumeration problems, even the problem of deciding the existence of one solution is not solvable in polynomial time, it

was proposed to allow FPT algorithms that have running time or delay $f(k) \cdot n^{O(1)}$ for some problem-specific parameter k [9, 11, 12, 14, 31]. Naturally, FPT-enumeration algorithms are based on extensions of standard techniques in FPT algorithms such as bounded-depth search trees [11, 12, 14] or color coding [31].

An important technique for obtaining FPT algorithms for decision problems is *kernelization* [10, 15, 28], where the idea is to shrink the input instance in polynomial time to an equivalent instance whose size depends only on the parameter k. In fact, a parameterized problem admits an FPT algorithm if and only if it admits a kernelization. It seems particularly intriguing to use kernelization for enumeration problems as a small kernel can be seen as a compact representation of the set of feasible solutions. The first notion of kernelization in the context of enumeration problems were the *full kernels* defined by Damaschke [11]. Informally, a full kernel for an instance of an enumeration problem is a subinstance that contains all minimal solutions of size at most k. This definition is somewhat restrictive since it is tied to subset minimization problems parameterized by the solution size parameter k. Nevertheless, full kernels have been obtained for some problems [12, 16, 26, 35].

To overcome the restrictions of full kernels, Creignou et al. [9] proposed *enumeration kernels*. Informally, an enumeration kernel for a parameterized enumeration problem is an algorithm that replaces the input instance by one whose size is bounded by the parameter and which has the property that the solutions of the original instance can be computed by listing the solutions of the kernel and using an efficient *solution-lifting algorithm* that outputs for each solution of the kernel a set of solutions of the original instance. In the definition of Creignou et al. [9], the solution-lifting algorithm may be an FPT-delay algorithm, that is, an algorithm with $f(k) \cdot n^{O(1)}$ delay where n is the overall input size. We find that this time bound is too weak, because it essentially implies that every enumeration problem that can be solved with FPT-delay admits an enumeration kernel of *constant* size. Essentially, this means that the solution-lifting algorithm is so powerful that it can enumerate all solutions while ignoring the kernel. Motivated by this observation and the view of kernels as compact representations of the solution set, we modify the original definition of enumeration kernels [9].

Our results. We present two new notions of efficient enumeration kernels by replacing the demand for FPT-delay algorithms by a demand for polynomial-time enumeration algorithms or polynomial-delay algorithms, respectively. We call the two resulting notions of enumeration kernelization *fully-polynomial enumeration kernels* and *polynomial-delay enumeration kernels*. Our paper aims at showing that these two new definitions present a sweet spot between the notion of full kernels, which is too strict for some applications, and enumeration kernels, which are too lenient in some sense. We first show that the two new definitions capture the class of efficiently enumerable problems in the sense that a problem has a fully-polynomial (a polynomial-delay) enumeration kernel if and only if it has an FPT-enumeration algorithm (an FPT-delay enumeration algorithm). Moreover, the kernels have constant size if and only if the problems have polynomial-time (polynomial-delay) enumeration algorithms. Thus, the new definitions correspond to the case of problem kernels for decision problems, which are in FPT if and only if they have kernels and which can be solved in polynomial time if and only if they have kernels of constant size (see, e.g. [10, Chapter 2] or [15, Chapter 1]).

We then apply both types of kernelizations to the enumeration of matching cuts. A matching cut of a graph G is the set of edges $M = E(A, B)$ for a partition $\{A, B\}$ of $V(G)$ forming a matching. We investigate the problems of enumerating all minimal, all maximal, or all matching cuts of a graph. We refer to these problems as Enum Minimal MC, Enum Maximal MC, and Enum MC, respectively. These matching cut problems constitute a

very suitable study case for enumeration kernels, since it is NP-hard to decide whether a graph has a matching cut [7] and therefore, they do not admit polynomial output-sensitive algorithms. We consider all three problems with respect to structural parameterizations such as the vertex cover number, the modular width, or the cyclomatic number of the input graph. The choice of these parameters is motivated by the fact that neither problem admits an enumeration kernel of polynomial size for the more general structural parameterizations by the treewidth or cliquewidth up to some natural complexity assumptions (see Proposition 5). Table 1 summarizes the results. Due to space constraints some results and proofs are either omitted or just sketched. We refer to the full version [20] for the details.

Table 1 An overview of our results. Herein, "kernel" means fully-polynomial enumeration kernel, "del-kernel" means polynomial-delay enumeration kernel and "bi" means bijective enumeration kernel (a slight generalization of full kernels), a $(\star)$ means that the lower bound assumes NP $\not\subseteq$ coNP/ poly, "?" means open status. We use $(*)$ for statements whose proofs are omitted in this extended abstract (see [20] for the proofs). The cyclomatic number is also known as the feedback edge number.

Parameter k	Enum MC	Enum Minimal MC	Enum Maximal MC
treewidth & cliquewidth	No poly-size del-kernel $(\star)$ (Prop. 5)	No poly-size del-kernel $(\star)$ (Prop. 5)	No poly-size del-kernel $(\star)$ (Prop. 5)
vertex cover & twin-cover number	size-$\mathcal{O}(k^2)$ del-kernel (Theorems 11 & 12) No kernel	size-$\mathcal{O}(k^2)$ kernel (Theorems 11 & 12)	size-$\mathcal{O}(k^2)$ del-kernel (Theorems 11 & 12) No kernel
neighborhood diversity	size-$\mathcal{O}(k)$ del-kernel $(*)$ No kernel $(*)$	size-$\mathcal{O}(k)$ kernel $(*)$	size-$\mathcal{O}(k)$ del-kernel $(*)$ No kernel $(*)$
modular width	?	$\mathcal{O}(k)$-kernel $(*)$	?
cyclomatic number	size-$\mathcal{O}(k)$ del-kernel (Theorem 13) No kernel	size-$\mathcal{O}(k)$ del-kernel (Theorem 13)	?
clique partition number	size-$\mathcal{O}(k^3)$ bi kernel $(*)$	size-$\mathcal{O}(k^3)$ bi kernel $(*)$	size-$\mathcal{O}(k^3)$ bi kernel $(*)$

To discuss some of our results and their implication for enumeration kernels in general more precisely, consider Enum MC, Enum Minimal MC, and Enum Maximal MC parameterized by the vertex cover number. We show that Enum Minimal MC admits a fully-polynomial enumeration kernel of polynomial size. As it can be seen that the problem has no full kernel, we obtain that there are natural enumeration problems with a fully-polynomial enumeration kernel that have no full kernel (not even one of super-polynomial size). Then, we show that Enum MC and Enum Maximal MC admit polynomial-delay enumeration kernels but have no fully-polynomial enumeration kernels. Thus, there are natural enumeration problems with polynomial-delay enumeration kernels that do not admit fully-polynomial enumeration kernels (not even one of super-polynomial size).

We also prove a tight upper bound $F(n+1)-1$ for the maximum number of matching cuts of an n-vertex graph, where $F(n)$ is the n-th Fibonacci number and show that all matching cuts can be enumerated in $\mathcal{O}^*(F(n)) = \mathcal{O}^*(1.6181^n)$ time (Theorem 6).

Related work. The current-best exact decision algorithm for Matching Cut, the problem of deciding whether a given graph G has a matching cut, has a running time of $O(1.328^n)$ where n is the number of vertices in G [25]. Faster exact algorithms can be obtained for the case when the minimum degree is large [23]. Matching Cut has FPT-algorithms for the maximum cut size k [21], the vertex cover number of G [27], and weaker parameters such as the twin-cover number [1] or the cluster vertex deletion number [25].

For an overview of enumeration algorithms, refer to the survey of Wasa [37]. A broader discussion of parameterized enumeration is given by Meier [32]. A different extension of enumeration kernels are *advice enumeration kernels* [2]. In these kernels, the solution-lifting algorithm does not need the whole input but only a possibly smaller advice. A further loosely connected extension of standard kernelization are *lossy kernels* which are used for optimization problems [29]; the common thread is that both definitions use a solution-lifting algorithm for recovering solutions of the original instance.

Graph notation. All graphs considered in this paper are finite undirected graphs without loops or multiple edges. We follow the standard graph-theoretic notation and terminology and refer to the book of Diestel [13] for basic definitions. For each of the graph problems considered in this paper, we let $n = |V(G)|$ and $m = |E(G)|$ denote the number of vertices and edges, respectively, of the input graph G if it does not create confusion. For a graph G and a subset $X \subseteq V(G)$ of vertices, we write $G[X]$ to denote the subgraph of G induced by X. For a set of vertices X, $G - X$ denotes the graph obtained by deleting the vertices of X, that is, $G - X = G[V(G) \setminus X]$; for a vertex v, we write $G - v$ instead of $G - \{v\}$. Similarly, for a set of edges A (an edge e, respectively), $G - A$ ($G - e$, respectively) denotes the graph obtained by the deletion of the edges of A (the edge e, respectively). For a vertex v, we denote by $N_G(v)$ the *(open) neighborhood* of v, i.e., the set of vertices that are adjacent to v in G. We use $N_G[v]$ to denote the *closed neighborhood* $N_G(v) \cup \{v\}$ of v. For $X \subseteq V(G)$, $N_G[X] = \bigcup_{v \in X} N_G[v]$ and $N_G(X) = N_G[X] \setminus X$. For disjoint sets of vertices A and B of a graph G, $E_G(A, B) = \{uv \mid u \in A,\ v \in B\}$. We may omit subscripts in the above notation if it does not create confusion. We use P_n, C_n, and K_n to denote the n-vertex path, cycle, and complete graph, respectively. We write $G + H$ to denote the *disjoint union* of G and H, and we use kG to denote the disjoint union of k copies of G.

In a graph G, a *cut* is a partition $\{A, B\}$ of $V(G)$, and we say that $E_G(A, B)$ is an *edge cut*. A *matching* is an edge set in which no two of the edges have a common end-vertex; note that we allow empty matchings. A *matching cut* is a (possibly empty) edge set being an edge cut and a matching. We underline that by our definition, a matching cut is a set of edges, as sometimes in the literature (see, e.g., [7, 22]) a matching cut is defined as a partition $\{A, B\}$ of the vertex set such that $E(A, B)$ is a matching. While the two variants of the definitions are equivalent, say when the decision variant of the matching cut problem is considered, this is not the case in enumeration and counting when we deal with disconnected graphs. For example, the empty graph on n vertices has $2^{n-1} - 1$ partitions $\{A, B\}$ which all correspond to exactly one matching cut in the sense of our definition, namely $M = \emptyset$. A matching cut M of G is *(inclusion) minimal* (*maximal*, respectively) if G has no matching cut $M' \subset M$ ($M' \supset M$, respectively). Notice that a disconnected graph has exactly one minimal matching cut which is the empty set.

2 Parameterized Enumeration and Enumeration Kernels

We use the framework for parameterized enumeration proposed by Creignou et al. [9]. An *enumeration problem* (over a finite alphabet Σ) is a tuple $\Pi = (L, \mathsf{Sol})$ such that

(i) $L \subseteq \Sigma^*$ is a decidable language,

(ii) $\mathsf{Sol}\colon \Sigma^* \to \mathcal{P}(\Sigma^*)$ is a computable function such that for every $x \in \Sigma^*$, $\mathsf{Sol}(x)$ is a finite set and $\mathsf{Sol}(x) \neq \emptyset$ if and only if $x \in L$.

Here, $\mathcal{P}(A)$ is used to denote the powerset of a set A. A string $x \in \Sigma^*$ is an *instance*, and $\mathsf{Sol}(x)$ is the set of solutions to instance x. A *parameterized enumeration problem* is defined as a triple $\Pi = (L, \mathsf{Sol}, \kappa)$ such that (L, Sol) satisfy (i) and (ii) of the above definition, and

(iii) $\kappa\colon \Sigma^* \to \mathbb{N}$ is a parameterization.

We say that $k = \kappa(x)$ is a *parameter*. We define the parameterization as a function of an instance but it is standard to assume that the value of $\kappa(x)$ is either simply given in x or can be computed in polynomial time from x. We follow this convention throughout the paper.

An *enumeration algorithm* $\mathcal{A}$ for a parameterized enumeration problem Π is a deterministic algorithm that for every instance x, outputs exactly the elements of $\mathsf{Sol}(x)$ without duplicates, and terminates after a finite number of steps on every instance. The algorithm $\mathcal{A}$ is an FPT *enumeration* algorithm if it outputs all solutions in at most $f(\kappa(x))p(|x|)$ steps for a computable function $f(\cdot)$ that depends only on the parameter and a polynomial $p(\cdot)$.

We also consider output-sensitive enumerations, and for this, we define delays. Let $\mathcal{A}$ be an enumeration algorithm for Π. For $x \in L$ and $1 \leq i \leq |\mathsf{Sol}(x)|$, the *$i$-th delay* of $\mathcal{A}$ is the time between outputting the i-th and $(i+1)$-th solutions in $\mathsf{Sol}(x)$. The 0-th delay is the *precalculation* time which is the time from the start of the computation until the output of the fist solution, and the $|\mathsf{Sol}(x)|$-th delay is the *postcalculation* time which is the time after the last output and the termination of $\mathcal{A}$ (if $\mathsf{Sol}(x) = \emptyset$, then the precalculation and postcalculation times are the same). It is said that $\mathcal{A}$ is a *polynomial-delay* algorithm, if all the delays are upper-bounded by $p(|x|)$ for a polynomial $p(\cdot)$. For a parameterized enumeration problem Π, $\mathcal{A}$ is an FPT*-delay algorithm*, if the delays are at most $f(\kappa(x))p(|x|)$, where $f(\cdot)$ is a computable function and $p(\cdot)$ is a polynomial. Notice that every FPT enumeration algorithm $\mathcal{A}$ is also an FPT delay algorithm.

The key definition for us is the generalization of the standard notion of a kernel in Parameterized Complexity (see, e.g, [15]) for enumeration problems.

▶ **Definition 1.** *Let $\Pi = (L, \mathsf{Sol}, \kappa)$ be a parameterized enumeration problem. A* fully-polynomial enumeration kernel(ization) *for Π is a pair of algorithms $\mathcal{A}$ and $\mathcal{A}'$ with the following properties:*

- **(i)** *For every instance x of Π, $\mathcal{A}$ computes in time polynomial in $|x| + \kappa(x)$ an instance y of Π such that $|y| + \kappa(y) \leq f(\kappa(x))$ for a computable function $f(\cdot)$.*
- **(ii)** *For every $s \in \mathsf{Sol}(y)$, $\mathcal{A}'$ computes in time polynomial in $|x| + |y| + \kappa(x) + \kappa(y)$ a nonempty set of solutions $S_s \subseteq \mathsf{Sol}(x)$ such that $\{S_s \mid s \in \mathsf{Sol}(y)\}$ is a partition of $\mathsf{Sol}(x)$.*

Notice that by (ii), $x \in L$ if and only if $y \in L$.

We say that $\mathcal{A}$ is a *kernelization* algorithm and $\mathcal{A}'$ is a *solution-lifting* algorithm. Informally, a solution-lifting algorithm takes as its input a solution for a "small" instance constructed by the kernelization algorithm and, having an access to the original input instance, outputs polynomially many solutions for the original instance, and by going over all the solutions to the small instance, we can generate all the solutions of the original instance without repetitions. We say that an enumeration kernel is *bijective* if $\mathcal{A}'$ produces a unique solution to x, that is, it establishes a bijection between $\mathsf{Sol}(y)$ and $\mathsf{Sol}(x)$, that is, the compressed instance essentially has the same solutions as the input instance. In particular, full kernels [11] are the special case of bijective kernels where $\mathcal{A}'$ is the identity. As it is standard, $f(\cdot)$ is the *size* of a kernel, and the kernel has *polynomial size* if $f(\cdot)$ is a polynomial.

We define *polynomial-delay enumeration kernel(ization)* in a similar way. The only difference is that (ii) is replaced by the condition

- **(ii*)** For every $s \in \mathsf{Sol}(y)$, $\mathcal{A}'$ computes with delay polynomial in $|x| + |y| + \kappa(x) + \kappa(y)$ a set of solutions $S_s \subseteq \mathsf{Sol}(x)$ such that $\{S_s \mid s \in \mathsf{Sol}(y)\}$ is a partition of $\mathsf{Sol}(x)$.

It is straightforward to make the following observation.

▶ **Observation 2.** *Every bijective enumeration kernel is a fully-polynomial enumeration kernel; every fully-polynomial enumeration kernel is a polynomial-delay enumeration kernel.*

Notice also that our definition of polynomial-delay enumeration kernel is different from the definition given by Creignou et al. [9]. In their definition, Creignou et al. [9] require that the solution-lifting algorithm $\mathcal{A}'$ should list all the solutions in S_s with FPT delay for the parameter $\kappa(x)$. We believe that this condition is too weak. In particular, with this requirement, every parameterized enumeration problem, that has an FPT enumeration algorithm $\mathcal{A}^*$ and such that the existence of at least one solution can be verified in polynomial time, has a trivial kernel of constant size. The kernelization algorithm can output any instance satisfying (i) and then we can use $\mathcal{A}^*$ as a solution-lifting algorithm that essentially ignores the output of the kernelization algorithm. Note that for enumeration problems, we typically face the situation where the existence of at least one solution is not an issue. We argue that our definitions are natural by showing the following theorem.

▶ **Theorem 3.** *A parameterized enumeration problem* Π *has an* FPT *enumeration algorithm (an* FPT *delay algorithm) if and only if* Π *admits a fully-polynomial enumeration kernel (polynomial-delay enumeration kernel). Moreover,* Π *can be solved in polynomial time (with polynomial delay) if and only if* Π *admits a fully-polynomial enumeration kernel (a polynomial-delay enumeration kernel) of constant size.*

Proof. The proof of the first claim is similar to the standard arguments for showing the equivalence between fixed-parameter tractability and the existence of a kernel (see, e.g. [10, Chapter 2] or [15, Chapter 1]). However dealing with enumeration problems requires some specific arguments. Let $\Pi = (L, \mathsf{Sol}, \kappa)$ be a parameterized enumeration problem.

In the forward direction, the claim is trivial. Recall that L is decidable and $\mathsf{Sol}(\cdot)$ is a computable function by the definition. If Π admits a fully-polynomial enumeration kernel (a polynomial-delay enumeration kernel respectively), then we apply an arbitrary enumeration algorithm, which is known to exist since $\mathsf{Sol}(\cdot)$ is computable, to the instance y produced by the kernelization algorithm. Then, for each $s \in \mathsf{Sol}(y)$, use the solution-lifting algorithm to list the solutions to the input instance.

For the opposite direction, assume that Π can be solved in $f(\kappa(x)) \cdot |x|^c$ time (with $f(\kappa(x)) \cdot |x|^c$ delay, respectively) for an instance x, where $f(\cdot)$ is a computable function and c is a positive constant. Since $f(\cdot)$ is computable, we assume that we have an algorithm $\mathcal{F}$ computing $f(k)$ in $g(k)$ time. We define $h(k) = \max\{f(k), g(k)\}$.

We say that an instance x of Π is a *trivial no-instance* if x is an instance of minimum size with $\mathsf{Sol}(x) = \emptyset$. We call x a *minimum yes-instance* if x is an instance of minimum size that has a solution. Notice that if Π has instances without solutions, then the size of a trivial no-instance is a constant that depends on Π only and such an instance can be computed in constant time. Similarly, if the problem has instances with solutions, then the size of a minimum yes-instance is constant and such an instance can be computed in constant time. We say that x is a *trivial yes-instance* if x is an instance with minimum size of $\mathsf{Sol}(x)$ that, subject to the first condition, has minimum size. Clearly, the size of a trivial yes-instance is a constant that depends only on Π. However, we may be unable to compute a trivial yes-instance.

Let x be an instance of Π and $k = \kappa(x)$. We run the algorithm $\mathcal{F}$ to compute $f(k)$ for at most $n = |x|$ steps. If the algorithm failed to compute $f(k)$ in n steps, we conclude that $g(k) \geq n$. In this case, the kernelization algorithm outputs x. Then the solution-lifting algorithm just trivially outputs its input solutions. Notice that $|x| \leq g(k) \leq h(k)$ in this case. Assume from now that $\mathcal{F}$ computed $f(k)$ in at most n steps.

If $|x| \leq f(k)$, then the kernelization algorithm outputs the original instance x, and the solution-lifting algorithm trivially outputs its input solutions. Note that $|x| \leq f(k) \leq h(k)$.

Finally, we suppose that $f(k) < |x|$. Observe that the enumeration algorithm runs in $|x|^{c+1}$ time (with $|x|^{c+1}$ delay, respectively) in this case, that is, the running time is polynomial. We use the enumeration algorithm to verify whether x has a solution. For this, notice that a polynomial-delay algorithm can be used to solve the decision problem; we just run it until it outputs a first solution (or reports that there are no solutions). If x has no solution, then Π has a trivial no-instance and the kernelization algorithm computes and outputs it. If x has a solution, then the kernelization algorithm computes a minimum yes-instance y in constant time. We use the enumeration algorithm to check whether $|\mathsf{Sol}(y)| \leq |\mathsf{Sol}(x)|$. If this holds, then we set $z = y$. Otherwise, if $|\mathsf{Sol}(x)| < |\mathsf{Sol}(y)|$, we find an instance z of minimum size such that $|\mathsf{Sol}(z)| \leq |\mathsf{Sol}(x)|$. Notice that this can be done in constant time, because the size of z is upper-bounded by the size of a trivial yes-instance. Then we list the solutions of z in constant time and order them. For the i-th solution of z, the solution-lifting algorithm outputs the i-th solution of x produced by the enumeration algorithm, and for the last solution of z, the solution-lifting algorithm further runs the enumeration algorithm to output the remaining solutions. Since $|\mathsf{Sol}(z)| \leq |\mathsf{Sol}(x)|$, the solution-lifting algorithm outputs a nonempty set of solutions for x for every solution of z.

It is easy to see that we obtain a fully-polynomial enumeration kernel of size $\mathcal{O}(h(\kappa(x))$ (a polynomial-delay enumeration kernel, respectively).

For the second claim, the arguments are the same. If a problem admits a fully-polynomial (a polynomial-delay) enumeration kernel of constant size, then the solutions of the original instance can be listed in polynomial time (or with polynomial delay, respectively) by the solution-lifting algorithm called for the constant number of the solutions of the kernel. Conversely, if a problem can be solved in polynomial time (with polynomial delay, respectively), we can apply the above arguments assuming that $f(k)$ (and, therefore, $g(k)$) is a constant. ◀

In our paper, we consider structural parameterizations of Enum Minimal MC, Enum Maximal MC, and Enum MC by several graph parameters, and the majority of these parameterizations are stronger than the parameterization either by the *treewidth* or the *cliquewidth* of the input graph. Defining the treewidth (denoted by $\mathsf{tw}(G)$) and cliquewidth (denoted by $\mathsf{cw}(G)$) goes beyond of the scope of the current paper and we refer to [8] (see also, e.g., [10]). By the celebrated result of Bodlaender [3] (see also [10]), it is FPT in t to decide whether $\mathsf{tw}(G) \leq t$ and to construct the corresponding tree-decomposition. No such algorithm is known for cliquewidth. However, for algorithmic purposes, it is usually sufficient to use the approximation algorithm of Oum and Seymour [34] (see also [33, 10]). Observe that the property that a set of edges M of a graph G is a matching cut of G can be expressed in *monadic second-order logic* (MSOL); we refer to [8, 10] for the definition of MSOL on graphs. Then the matching cuts (the minimal or maximal matching cuts) of a graph of treewidth at most t can be enumerated with FPT delay with respect to the parameter t by the celebrated meta theorem of Courcelle [8]. The same holds for the weaker parameterization by the cliquewidth of the input graph, because we can use MSOL formulas without quantifications over (sets of) edges: For a graph G, we pick a vertex in each connected component of G and label it. Let R be the set of labeled vertices. Then the enumeration of nonempty matching cuts is equivalent to the enumeration of all partitions $\{A, B\}$ of $V(G)$ such that (i) $R \subseteq A$ and (ii) $E(A, B)$ is a matching. Notice that condition (ii) can be written as follows: for every $u_1, u_2 \in A$ and $v_1, v_2 \in B$, if u_1 is adjacent to v_1 and u_2 is adjacent to v_2, then either $u_1 = u_2$ and $v_1 = v_2$ or $u_1 \neq u_2$ and $v_1 \neq v_2$. Since the empty matching cut can be listed separately if it exists, we obtain that we can use MSOL formulations of the enumeration problems, where only quantifications over vertices and sets of vertices are used. Then the result of Courcelle [8] implies that Enum Minimal MC, Enum Maximal MC, and Enum MC can be solved with FPT delay when parameterized by the cliquewidth of the input graph.

We summarize these observations in the following proposition.

▶ **Proposition 4.** *Enum MC, Enum Minimal MC, and Enum Maximal MC on graphs of treewidth (cliquewidth) at most t can be solved with* FPT *delay when parameterized by t.*

This proposition implies that Enum MC, Enum Minimal MC and Enum Maximal MC can be solved with FPT delay for all structural parameters whose values can be bounded from below by an increasing function of treewidth or cliquewidth. However, we are mainly interested in fully-polynomial or polynomial-delay enumeration kernelization. We conclude this section by pointing out that it is unlikely that Enum Minimal MC, Enum Maximal MC, and Enum MC admit polynomial-delay enumeration kernels of polynomial size for the treewidth or cliquewidth parameterizations. It was pointed out by Komusiewicz, Kratsch, and Le [25] that the decision version of the matching cut problem (that is, the problem asking whether a given graph G has a matching cut) does not admit a polynomial kernel when parameterized by the treewidth of the input graph unless NP $\subseteq$ coNP/ poly. By the definition of a polynomial-delay enumeration kernel, this gives the following statement.

▶ **Proposition 5.** *Enum Minimal MC, Enum Maximal MC and Enum MC do not admit polynomial-delay enumeration kernels of polynomial size when parameterized by the treewidth (cliquewidth, respectively) of the input graph unless* NP $\subseteq$ coNP/ poly*.*

3 A Tight Upper Bound for the Maximum Number of Matching Cuts

In this section we provide a tight upper bound for the maximum number of matching cuts of an n-vertex graph. We complement this result by giving an exact enumeration algorithm for (minimal, maximal) matching cuts. Finally, we give some lower bounds for the maximum number of minimal and maximal matching cuts, respectively. Throughout this section, we use $\#_{\mathrm{mc}}(G)$ to denote the number of matching cuts of a graph G.

To give the upper bound, we use the classical Fibonacci numbers. For a positive integer n, we denote by $F(n)$ the n-th Fibonacci number. Recall that $F(1) = F(2) = 1$, and for $n \geq 3$, the Fibonacci numbers satisfy the recurrence $F(n) = F(n-1) + F(n-2)$. Recall also that the n-th Fibonacci number can be expressed by the following closed formula:

$$F(n) = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n + \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

for every $n \geq 1$. In particular, $F(n) = \mathcal{O}(1.6181^n)$.

▶ **Theorem 6** $(*)$**.** [1] *An n-vertex graph has at most $F(n+1)-1$ matching cuts. The bound is tight and is achieved for paths. Moreover, if $n \geq 5$, then an n-vertex graph G has $F(n+1)-1$ matching cuts if and only if G is a path. Furthermore, the matching cuts can be enumerated in $\mathcal{O}^*(F(n))$ time.*

Let us remark that if $n \leq 4$, then besides paths P_n, the graphs $K_p + K_q$ for $1 \leq p, q \leq 2$ such that $n = p + q$ have $F(n+1) - 1$ matching cuts.

Clearly, the upper bound for the maximum number of matching cuts given in Theorem 6 is an upper bound for the maximum number of minimal and maximal matching cuts. However, the number of minimal or maximal matching cuts may be significantly less than the number of all matching cuts. We conclude this section by stating the best lower bounds we know for the maximum number of maximal matching cuts and minimal matching cuts, respectively.

[1] The proofs of the statements labeled $(*)$ are omitted in this extended abstract.

▶ **Proposition 7** (∗). *The graph $G = kC_7$ with $n = 7k$ vertices has $14^k = 14^{n/7} \geq 1.4579^n$ maximal matching cuts.*

To achieve a lower bound for the maximum number of minimal matching cuts, we consider the graphs H_k constructed as follows for a positive integer k.

- For every $i \in \{1, \ldots, k\}$, construct two vertices u_i and v_i and a (u_i, v_i)-path of length 4.
- Make the vertices $u_1, \ldots, u_k$ pairwise adjacent, and do the same for $v_1, \ldots, v_k$.

▶ **Proposition 8** (∗). *The number of minimal matching cuts of H_k with $n = 5k$ vertices is at least $4^k = 4^{n/5} \geq 1.3195^n$.*

4 Enumeration Kernels for the Vertex Cover Number Parameterization

In this section, we consider the parameterization of the matching cut problems by the vertex cover number of the input graph. Notice that this parameterization is one of the most thoroughly investigated with respect to classical kernelization (see, e.g., the recent paper of Bougeret, Jansen, and Sau [6] for the currently most general results of this type). However, we are interested in enumeration kernels.

Recall that a set of vertices $X \subseteq V(G)$ is a *vertex cover* of G if for every edge $uv \in E(G)$, at least one of its end-vertices is in X, that is, $V(G) \setminus X$ is an independent set. The *vertex cover number* of G, denoted by $\tau(G)$, is the minimum size of a vertex cover of G. Computing $\tau(G)$ is NP-hard but one can find a 2-approximation by taking the end-vertices of a maximal matching of G [19] (see also [24] for a better approximation) and this suffices for our purposes. Throughout this section, we assume that the parameter $k = \tau(G)$ is given together with the input graph. Note that for every graph G, $\mathsf{tw}(G) \leq \tau(G)$. Therefore, Enum MC, Enum Minimal MC, and Enum Maximal MC can be solved with FPT delay when parameterized by the vertex cover number by Proposition 4.

First, we describe the basic kernelization algorithm that is exploited for all the kernels in this subsection. Let G be a graph that has a vertex cover of size k. The case when G has no edges is trivial and will be considered separately. Assume from now that G has at least one edge and $k \geq 1$.

We use the above-mentioned 2-approximation algorithm to find a vertex cover X of size at most $2k$. Let $I = V(G) \setminus X$. Recall that I is an independent set. Denote by I_0, I_1, and $I_{\geq 2}$ the subsets of vertices of I of degree 0, 1, and at least 2, respectively. We use the following *marking* procedure to label some vertices of I.

- **(i)** *Mark* an arbitrary vertex of I_0 (if it exists).
- **(ii)** For every $x \in X$, *mark* an arbitrary vertex of $N_G(x) \cap I_1$ (if it exists).
- **(iii)** For every two distinct vertices $x, y \in X$, select an arbitrary set of $\min\{3, |(N_G(x) \cap N_G(y)) \cap I_{\geq 2}|\}$ vertices in $I_{\geq 2}$ that are adjacent to both x and y, and *mark* them for the pair $\{x, y\}$.

Note that a vertex of $I_{\geq 2}$ can be marked for distinct pairs of vertices of X. Denote by Z the set of marked vertices of I. Clearly, $|Z| \leq 1 + |X| + 3\binom{|X|}{2}$. We define $H = G[X \cup Z]$. Notice that $|V(H)| \leq |X| + |Z| \leq 1 + 2|X| + 3\binom{|X|}{2} \leq 6k^2 + k + 1$. This completes the description of the basic kernelization algorithm that returns H. It is straightforward to see that H can be constructed in polynomial time.

It is easy to see that H does not keep the information about all matching cuts in G due to the deleted vertices. However, the crucial property is that H keeps all matching cuts of $G' = G - (I_0 \cup I_1)$. Formally, we define $H' = H - (I_0 \cup I_1)$ and show the following lemma.

▶ **Lemma 9** (∗). *A set of edges $M \subseteq E(G')$ is a matching cut of G' if and only if $M \subseteq E(H')$ and M is a matching cut of H'.*

To see the relations between matching cuts of G and H, we define a special equivalence relation for the subsets of edges of G. For a vertex $x \in X$, let $L_x = \{xy \in E(G) \mid y \in I_1\}$, that is, L_x is the set of pendant edges of G with exactly one end-vertex in the vertex cover. Observe that if $L_x \neq \emptyset$, then there is $\ell_x \in L_x$ such that $\ell_x \in E(H)$, because for every $x \in X$, a neighbor in I_1 is marked if it exists. We define $L = \bigcup_{x \in X} L_x$. Notice that each matching cut of G contains at most one edge of every L_x. We say that two sets of edges M_1 and M_2 are *equivalent* if $M_1 \setminus L = M_2 \setminus L$ and for every $x \in X$, $|M_1 \cap L_x| = |M_2 \cap L_x|$. It is straightforward to verify that the introduced relation is indeed an equivalence relation. It is also easy to see that if M is a matching cut of G, then every $M' \subseteq E(G)$ equivalent to M is a matching cut. We show the following lemma.

▶ **Lemma 10** (∗). *A set of edges $M \subseteq E(G)$ is a matching cut (minimal or maximal matching cut, respectively) of G if and only if H has a matching cut (minimal or maximal matching cut, respectively) M' equivalent to M.*

We use Lemma 10 to obtain our kernelization results. For Enum Minimal MC, we show that the problem admits a fully-polynomial enumeration kernel, and we prove that Enum Maximal MC and Enum MC have polynomial-delay enumeration kernels.

▶ **Theorem 11.** *Enum Minimal MC admits a fully-polynomial enumeration kernel and Enum MC and Enum Maximal MC admit polynomial-delay enumeration kernels with $\mathcal{O}(k^2)$ vertices when parameterized by the vertex cover number k of the input graph.*

Proof. Let G be a graph with $\tau(G) = k$. If $G = K_1$, then the kernelization algorithm returns $H = G_1$ and the solution-lifting algorithm is trivial as G has no matching cuts. Assume that G has at least 2 vertices. If G has no edges, then the empty set is the unique matching cut of G. Then the kernelization algorithm returns $H = 2K_1$, and the solution-lifting algorithm outputs the empty set for the empty matching cut of H. Thus, we can assume without loss of generality that G has at least one edge and $k \geq 1$.

We use the same basic kernelization algorithm that constructs H as described above and output H for all the problems. Recall that $|V(H)| \leq 6k^2+k+1$. The kernels differ only in the solution-lifting algorithms. These algorithms exploit Lemma 10 and for every matching cut (minimal or maximal matching cut, respectively) M of H, they list the equivalent matching cuts of G. Lemma 10 guarantees that the families of matching cuts (minimal or maximal matching cuts, respectively) constructed for every matching cut of H compose the partition of the sets of matching cuts (minimal or maximal matching cuts, respectively) of G. This is exactly the property that is required by the definition of a fully-polynomial (polynomial-delay) enumeration kernel. To describe the algorithm, we use the notation defined in this section.

First, we consider Enum Minimal MC. Let M be a minimal matching cut of H. If $M \cap L = \emptyset$, then M is the unique matching cut of G that is equivalent to M, and our algorithm outputs M. Suppose that $M \cap L \neq \emptyset$. Then by the minimality of M, $M = \{\ell_x\}$ for some $x \in X$, because every edge of L is a matching cut. Then the sets $\{e\}$ for every $e \in L_x$ are exactly the matching cuts equivalent to M. Clearly, we have at most n such matching cuts and they can be listed in linear time. This implies that condition (ii) of the definition of a fully-polynomial enumeration kernel is fulfilled. Thus, Enum Minimal MC has a fully-polynomial enumeration kernel with at most $6k^2 + k + 1$ vertices.

Next, we consider Enum Maximal MC and Enum MC. The solution-lifting algorithms for these problems are the same. Let M be a (maximal) matching cut of H. Let also $M_1 = M \cap L$ and $M_2 = M \setminus M_1$. If $M_1 = \emptyset$, then M is the unique matching cut of G that is

equivalent to M, and our algorithm outputs M. Assume from now that $M_1 \neq \emptyset$. Then there is $Y \subseteq X$ such that $M_1 = \{\ell_x \mid x \in Y\}$. We use the recursive algorithm Enum Equivalent (see Algorithm 1) that takes as an input a matching S of G and $W \subseteq Y$ and outputs the equivalent matching cuts M' of G such that (i) $S \subseteq M'$, (ii) M' is equivalent to M, and (iii) the constructed matchings M' differ only by some edges of the sets L_x for $x \in W$. Initially, $S = M_2$ and $W = Y$.

Algorithm 1 Enum Equivalent(S, W).

```
if W = ∅ then
    output S
end
else if S ≠ ∅ then
    select arbitrary x ∈ W;
    foreach e ∈ L_x do
        Enum Equivalent(S ∪ {e}, W \ {x})
    end
end
```

To enumerate the matching cuts equivalent to M, we call Enum Equivalent(M_2, Y). We claim that Enum Equivalent(M_2, Y) enumerates the matching cuts of G that are equivalent to M with $\mathcal{O}(n)$ delay.

By the definition of the equivalence and Lemma 10, every matching cut M' of G that is equivalent to M can be written as $M' = M_2 \cup \{e_x \mid x \in Y\}$, where e_x is an edge of L_x for $x \in Y$. Then to see the correctness of Enum Equivalent, observe the following. If $W \neq \emptyset$, then the algorithm picks a vertex $x \in W$. Then for every edge $e \in L_x$, it enumerates the matching cuts containing S and e. This means that our algorithm is, in fact, a standard *backtracking* enumeration algorithm (see [30]) and immediately implies that the algorithm lists all the required matching cuts exactly once. Since the depth of the recursion is at most n and the algorithm always outputs a matching cut for each leaf of the search tree, the delay is $\mathcal{O}(n)$. This completes the proof of the polynomial-delay enumeration kernel for Enum Maximal MC and Enum MC.

To conclude the proof of the theorem, let us remark that, formally, the solution-lifting algorithms described in the proof require X. However, in fact, we use only sets L_x that can be computed in polynomial time for given G and H. ◀

Notice that Theorem 11 is tight in the sense that Enum Maximal MC and Enum MC do not admit fully-polynomial enumeration kernels for the parameterization by the vertex cover number. To see this, let k be a positive integer and consider the n-vertex graph G, where $n > k$ is divisible by k, that is the union of k stars $K_{1,p}$ for $p = n/k - 1$. Clearly, $\tau(G) = k$. We observe that G has $p^k = (n/k - 1)^k$ maximal matching cut that are formed by picking one edge from each of the k stars. Similarly, G has $(p+1)^k = (n/k)^k$ matching cuts obtained by picking at most one edge from each star. In both cases, this means that the (maximal) matching cuts cannot be enumerated by an FPT algorithm. By Theorem 3, this rules out the existence of a fully-polynomial enumeration kernel.

We conclude this section by showing that Theorem 11 can be generalized to the weaker parameterization by the *twin-cover* number, introduced by Ganian [17, 18] as a generalization of a vertex cover. Recall that two vertices u and v of a graph G are *true twins* if $N[u] = N[v]$. A set of vertices X of a graph G is said to be a *twin-cover* of G if for every edge uv of G,

at least one of the following holds: (i) $u \in X$ or $v \in X$ or (ii) u and v are true twins. The *twin-cover* number, denoted by $\mathsf{tc}(G)$, is the minimum size of a twin-cover. Notice that $\mathsf{tc}(G) \leq \tau(G)$ and $\mathsf{tc}(G) \geq \mathsf{cw}(G) + 2$ for every G [17, 18]. Let $\mathcal{X} = \{X_1, \ldots, X_r\}$ be the partition of $V(G)$ into the classes of true twins. Note that $\mathcal{X}$ can be computed in linear time using an algorithm for computing a modular decomposition [36]. Then we can define the *true-twin quotient* graph $\mathcal{G}$ with respect to $\mathcal{X}$, that is, the graph with the node set $\mathcal{X}$ such that two classes of true twins X_i and X_j are adjacent in $\mathcal{G}$ if and only if the vertices of X_i are adjacent to the vertices of X_j in G. Then it can be seen that $\mathsf{tc}(G) \geq \tau(\mathcal{G})$. We prove the following.

▶ **Theorem 12** (∗). *Enum Minimal MC admits a fully-polynomial enumeration kernel and Enum MC and Enum Maximal MC admit polynomial-delay enumeration kernels with $\mathcal{O}(k^2)$ vertices when parameterized by the vertex cover number of the true-twin quotient graph of the input graph.*

5 Enumeration Kernels for the Parameterization by the Feedback Edge Number

A set of edges X of a graph G is said to be a *feedback* edge set if $G - S$ has no cycle, that is, $G - S$ is a forest. The minimum size of a feedback edge set is called the *feedback edge* number or the *cyclomatic* number. We use $\mathsf{fn}(G)$ to denote the feedback edge number of a graph G. It is well-known (see, e.g., [13]) that if G is a graph with n vertices, m edges and r connected components, then $\mathsf{fn}(G) = m - n + r$ and a feedback edge set of minimum size can be found in linear time. Throughout this section, we assume that the input graph in an instance of Enum Minimal MC or Enum MC is given together with a feedback edge set. Equivalently, we may assume that kernelization and solution-lifting algorithms are supplied by the same algorithm computing a minimum feedback edge set. Then this algorithm computes exactly the same set for the given input graph.

In contrast to vertex cover number and neighborhood diversity, Enum Minimal MC does not admit a fully-polynomial enumeration kernel in case of the feedback edge number: let ℓ and k be positive integers and consider the graph $H_{k,\ell}$ that is constructed as follows.

- For every $i \in \{1, \ldots, k\}$, construct two vertices u_i and v_i and a (u_i, v_i)-path of length ℓ.
- Add edges to make each of $u_1, \cdots, u_k$ and $v_1, \cdots, v_k$ a path of length $k - 1$.

Observe that $H_{k,\ell}$ has at least ℓ^k minimal matching cuts composed by taking one edge from every (u_i, v_i)-path. Since $H_{k,\ell}$ has $n = k(\ell + 1)$ vertices and $\mathsf{fn}(H_{k,\ell}) = k - 1$, the number of minimal matching cuts is at least $\left(\frac{n}{\mathsf{fn}(H_{k,\ell})-1} - 1\right)^{\mathsf{fn}(H_{k,\ell})}$. This immediately implies that the minimal matching cuts cannot be enumerated in FPT time. In particular, Enum Minimal MC cannot have a fully-polynomial enumeration kernel by Theorem 3. However, this problem and Enum MC admit polynomial-delay enumeration kernels.

▶ **Theorem 13.** *Enum Minimal MC and Enum MC admit a polynomial-delay enumeration kernel with $\mathcal{O}(k)$ vertices when parameterized by the feedback edge number k of the input graph.*

The kernels for Enum Minimal MC and Enum MC are similar but the kernel for Enum MC requires some technical details that do not appear in the kernel for Enum Minimal MC. For Enum MC, we need the following observation that follows from the results of Courcelle [8] in the same way as Proposition 4 using a *Counting* MSOL formulation of the enumeration problem.

▶ **Observation 14.** *Let F be a forest and let $A, B, C \subseteq E(F)$ be disjoint edge sets. Then all matchings M of F such that $A \subseteq M$, $B \cap M = \emptyset$, and either $C \subseteq M$ or $C \cap M = \emptyset$ can be enumerated with polynomial delay. Moreover, if u, v are distinct vertices of the same connected component of F and $h \in \{0, 1\}$, then all such (nonempty) matchings with the additional property that $|E(P) \cap A| \mod 2 = h$, where P is the (u, v)-path of F, also can be enumerated with polynomial delay.*

Proof of Theorem 13. We sketch the proof for Enum MC. Let G be a graph with $\mathsf{fn}(G) = k$ and a feedback edge set S of size k. The case where G is a forest can be settled by using Observation 14 (or Proposition 4). We assume from now that G is not a forest. In particular, $S \neq \emptyset$. If G has one or more connected component that are trees, we select an arbitrary vertex v^* of these components. If G has a connected component that contains a vertex of degree one and is not a tree, then arbitrary select such a vertex u^* of degree one and denote by e^* be the edge incident to u^*. Then we iteratively delete vertices of degree at most one distinct from u^* and v^*. Denote by G' the obtained graph. Notice that G' has at most one isolated vertex (the vertex v^*) and at most one vertex of degree one (the vertex u^*). Observe also that S is a minimum feedback edge set of G'. Let $T = G' - S$. Notice that T is a forest and has at most $2|S| + 2 \leq 2k + 2$ vertices of degree at most one. It can be shown that T has at most $2k$ vertices of degree at least three. Denote by X the set of vertices of T that either are end-vertices of the edges of S, or have degree one, or have degree at least three. Then $|X| \leq 4k + 2$, and every vertex v of G' of degree two is an inner vertex of an (x, y)-path P such that $x, y \in X$ and the inner vertices of P are outside X. Moreover, for every two distinct $x, y \in X$, G' has at most one (x, y)-path P_{xy} with all its inner vertices outside X. We denote by $\mathcal{P}$ the set of all such paths. We say that an edge of P_{xy} is the *x-edge* if it is incident to x and is the *y-edge* if it is incident to y. We say that an edge e of P_{xy} is a *second x-edge* (a *second y-edge*, respectively) if e has a common end-vertex with the x-edge (with the y-edge, respectively). The edges that are distinct from the x-edge, the second x-edge, the y-edge and the second y-edge are called *middle edges*. We say that P_{xy} is *long* if P_{xy} has length at least six; otherwise, P_{xy} is *short*. Let $F = G - E(G')$. Since $S \subseteq E(G')$, F is a forest. Moreover, each connected component T of F has at most one vertex in $V(G')$.

We exhaustively apply the following reduction rule.

▶ **Reduction Rule.** *If there is a long path $P_{xy} \in \mathcal{P}$ for some $x, y \in X$, then contract an arbitrary middle edge of P_{xy}.*

Let H be the graph obtained from G' by the exhaustive application of the reduction rule. We also denote by $\mathcal{P}'$ the set of paths obtained from the paths of $\mathcal{P}$; we use P'_{xy} to denote the path obtained from $P_{xy} \in \mathcal{P}$. Our kernelization algorithm returns H together with S. It can be seen that $|V(H)| \leq 20k + 1$.

For the construction of the solution-lifting algorithm, recall that by our assumption the input graph is given together with S and $S \subseteq E(H)$. Then we can identify v^*, u^* and e^* in G and H, and then we can recompute the set X. Next, we can compute the sets of paths $\mathcal{P}$ and $\mathcal{P}'$ of G and H, respectively, in polynomial time. This allows us to assume that the solution-lifting algorithm has access to these sets.

To construct the solution-lifting algorithm, denote by $\mathcal{M}$ and $\mathcal{M}'$ the sets of matching cuts of G and H, respectively. Define $\mathcal{M}_1 = \{M \in \mathcal{M} \mid M \cap E(G') = \emptyset\}$ and $\mathcal{M}_2 = \{M \in \mathcal{M} \mid M \cap E(G') \neq \emptyset\}$. Notice that $M \in \mathcal{M}_1$ is nonempty if and only if M is a nonempty matching of $F = G - E(G')$. First, we deal with the matching cuts of $\mathcal{M}_1$. Observe that G is connected if and only if H is connected. This means that the empty set is a matching cut of G if and only if the empty set is a matching cut of H.

Suppose that H has the empty matching cut. Then the solution-lifting algorithm, given this matching cut of H, outputs the matching cuts of $\mathcal{M}_1$. Notice that $\mathcal{M}_1 \neq \emptyset$, because $\mathcal{M}_1$ contains the empty matching cut. The solution-lifting algorithm outputs the empty matching cut and all nonempty matchings of F using Observation 14.

Assume now that H is connected. Then G is connected as well and $\mathcal{M}_1 \neq \emptyset$ if and only if $F \neq \emptyset$. By the construction of G', if F is not empty, then G has a vertex of degree one. In particular, the kernelization algorithm selects u^* and e^* in this case. Notice that e^* is a bridge of G, and it holds that $\{e^*\}$ is a matching cut of both G and H. Observe also that $\{e^*\} \in \mathcal{M}_2$. This matching cut is generated by the solution-lifting algorithm for the cut $\{e^*\}$ of H: when the algorithm finishes listing the matching cuts of $\mathcal{M}_2$ for $\{e^*\}$, it switches to the listing of all nonempty matchings of F. This can be done with polynomial delay by Observation 14.

Next, we analyze the matching cuts of $\mathcal{M}_2$. By definition, a matching cut M of G is in $\mathcal{M}_2$ if $M \cap E(G') \neq \emptyset$. This means that $M \cap E(G')$ is a matching cut of G', and for a nonempty matching M of G, $M \in \mathcal{M}_2$ if and only if $M \cap E(G')$ is a nonempty matching cut of G'. We exploit this property and the solution-lifting algorithm lists nonempty matching cuts of G' and then for each matching cut of G', it outputs all its possible extensions by matchings of F. For this, we define the following relation between matching cuts of H and G'. Let M be a nonempty matching cut of H and let M' be a nonempty matching of G' (note that we do not require M' to be a matching cut). We say that M' is *equivalent* to M if the following holds:

- **(i)** $M \cap E(H[X]) = M' \cap E(G[X])$ (note that $H[X] = G[X]$).
- **(ii)** For every $P_{xy} \in \mathcal{P}$ such that P_{xy} is short, $M \cap E(P'_{xy}) = M' \cap E(P_{xy})$ (note that $P_{xy} = P'_{xy}$ in this case).
- **(iii)** For every long $P_{xy} \in \mathcal{P}$,
  - **(a)** $M \cap E(P'_{xy}) \neq \emptyset$ if and only if $M' \cap E(P_{xy}) \neq \emptyset$,
  - **(b)** $|M \cap E(P'_{xy})| \mod 2 = |M' \cap E(P_{xy})| \mod 2$,
  - **(c)** the x-edge (y-edge, respectively) of P'_{xy} is in M' if and only if the x-edge (y-edge, respectively) of P_{xy} is in M,
  - **(d)** if for the second x-edge e_x, the second y-edge e_y and the middle edge e of P'_{xy}, $|M \cap \{e_x, e_y, e\}| = 1$, then
    - $e_x \in M$ ($e_y \in M$, respectively) if and only if $e_x \in M'$ and $e_y \notin M'$ ($e_x \notin M'$ and $e_y \in M'$, respectively),
    - $e \in M$ if and only if either $e_x, e_y \in M'$ or $e_x, e_y \notin M'$.

    (note that e_x, e_y are the second x-edge and y-edge of P_{xy}, because P'_{xy} is constructed by contracting of some middle edges of P_{xy}).

We use the properties of the relation summarized in the following claim.

▷ Claim 15.

- **(i)** For every nonempty matching cut M of H, there is a nonempty matching M' of G' that is equivalent to M.
- **(ii)** For every nonempty matching cut M of H and every nonempty matching M' of G' equivalent to M, M' is a matching cut of G'.
- **(iii)** Every nonempty matching cut M' of G' is equivalent to at most one matching cut of H.
- **(iv)** For every nonempty matching cut M' of G', there is a nonempty matching cut of M such that M' is equivalent to M.

Claim 15 allows us to construct the solution-lifting algorithm for nonempty matching cuts of H that outputs nonempty matching cuts from $\mathcal{M}_2$. For each nonempty matching cut M of H, the algorithm lists the matching cuts M' of G' such that M' is equivalent to M. Then for each M', we extend M' to matching cuts of G by adding matchings of $F = G - E(G')$. For this, we consider the algorithm EnumPath$(P_{x,y}, A, B, C, h)$ that given a path $P_{xy} \in \mathcal{P}$, disjoint sets $A, B, C \subseteq E(P_{xy})$, and an integer $h \in \{0, 1\}$, enumerates with polynomial delay all nonempty matchings M of P_{xy} such that $A \subseteq M$, $B \cap M = \emptyset$, either $C \subseteq M$ or $C \cap M = \emptyset$, and $|M| \mod 2 = h$. Such an algorithm exists by Observation 14. We also use the algorithm EnumMatchF(M) that, given a matching cut M of G', lists all matching cuts of G of the form $M \cup M'$, where M' is a matching of F. EnumMatchF(M) is constructed as follows. Let A be the set of edges of F incident to the end-vertices of F (recall that each connected component of F contains at most one vertex of $V(G')$). Then we enumerate the matchings M' of F such that $M' \cap A = \emptyset$. This can be done with polynomial delay by Observation 14.

Algorithm 2 EnumEquivalent$(L, \mathcal{R})$.

if $\mathcal{R} = \emptyset$ **then**
  call EnumMatchF(M);
  return every matching cut M' generated by the algorithm and **quit**
end
else if $\mathcal{R} \neq \emptyset$ **then**
  select arbitrary $P_{xy} \in \mathcal{R}$;
  set $A := \emptyset$; $B := \emptyset$; $C := \emptyset$; $h := |M \cap E(P'_{xy})| \mod 2$;
  if $e_x \in M$ **then** set $A := A \cup \{e_x\}$;
  if $e_y \in M$ **then** set $A := A \cup \{e_y\}$;
  if $e'_x \in M$ *and* $e, e'_y \notin M$ **then** set $A := A \cup \{e'_x\}$ and $B := B \cup \{e'_y\}$;
  if $e'_y \in M$ *and* $e, e'_x \notin M$ **then** set $A := A \cup \{e'_y\}$ and $B := B \cup \{e'_x\}$;
  if $e \in M$ *and* $e'_x, e'_y \notin M$ **then** set $C := C \cup \{e'_x, e'_y\}$;
  call EnumPath$(P_{x,y}, A, B, C, h)$;
  foreach *nonempty matching* Z *generated by* EnumPath$(P_{x,y}, A, B, C, h)$ **do**
    EnumEquivalent$(L \cup Z, \mathcal{R} \setminus \{P_{xy}\})$
  end
end

We use EnumPath and EnumMatchF as subroutines of the recursive branching algorithm EnumEquivalent (see Algorithm 2) that, given a matching M of H, takes as an input a matching L of G and $\mathcal{R} \subseteq \mathcal{P}$ and outputs the matching cuts M' of G such that (i) $L \subseteq M'$, (ii) M' is equivalent to M, and (iii) the constructed matchings M' differ only by some edges of the paths $P_{xy} \in \mathcal{R}$. To initiate the computations, we construct the initial matching L' of G and the initial set of paths $\mathcal{R}' \subseteq \mathcal{P}$ as follows. We define $\mathcal{R}' \subseteq \mathcal{P}$ to be the set of long paths $P_{xy} \subseteq \mathcal{P}$ such that $P'_{xy} \cap M \neq \emptyset$. Then $L' \subseteq M$ is the set of edges of M that are not in the paths of $\mathcal{R}'$. Recall that as an intermediate step, we enumerate nonempty matching cuts of G' that are equivalent to M. Then it can be noted that to do this, we have to enumerate all possible extensions of M to M' satisfying condition (iii) of the equivalence definition. Therefore, we call EnumEquivalent$(L', \mathcal{R}')$ to solve the enumeration problem. It can be seen that EnumEquivalent$(L', \mathcal{R}')$ enumerates with polynomial delay all nonempty matching cuts $M \in \mathcal{M}_2$ such that $M' \cap E(G')$ is a nonempty matching cut of G' equivalent to M.

To summarize, recall that if H is connected and has a vertex of degree one, we used the matching cut $\{e^*\}$ to list the matching cuts formed by the edges of $F = G - E(G')$. Clearly, $\{e^*\}$ is generated by EnumEquivalent$(L', \mathcal{R}')$ for L' and $\mathcal{R}'$ constructed for $M = \{e^*\}$. Therefore, we conclude that the solution-lifting algorithm satisfies condition (ii*) of the definition of a polynomial-delay enumeration kernel. ◀

6 Conclusion

We initiated the systematic study of enumeration kernelization for several variants of the matching cut problem. We obtained fully-polynomial (polynomial-delay) enumeration kernels for the parameterizations by the vertex cover number, twin-cover number, neighborhood diversity, modular width, and feedback edge number. Since the solution-lifting algorithms are simple branching algorithms, these kernels give a condensed view of the solution sets which may be interesting in applications where one may want to inspect all solutions manually. Restricting to polynomial-time and polynomial-delay solution-lifting algorithms seems helpful in the sense that they will usually be easier to understand.

There are many topics for further research in enumeration kernelization. For Matching Cut, it would be interesting to investigate other structural parameters, like the feedback vertex number (see [10] for the definition). More generally, the area of enumeration kernelization seems still somewhat unexplored. It would be interesting to see applications of the various kernel types to other enumeration problems. For this, it seems to be important to develop general tools for enumeration kernelizations. For example, is it possible to establish a framework for enumeration kernelization lower bounds similar to the techniques used for classical kernels [4, 5] (see also [10, 15])?

Concerning the counting and enumeration of matching cuts, we also proved the upper bound $F(n+1) - 1$ for the maximum number of matching cuts of an n-vertex graph and showed that the bound is tight. What can be said about the maximum number of minimal and maximal matching cuts? It is not clear whether our lower bounds given in Propositions 7 and 8 are tight. Finally, it seems promising to study enumeration kernels for d-Cut [21], a generalization of Matching Cut that has recently received some attention.

References

1 N. R. Aravind, Subrahmanyam Kalyanasundaram, and Anjeneya Swami Kare. On structural parameterizations of the matching cut problem. In Xiaofeng Gao, Hongwei Du, and Meng Han, editors, *Combinatorial Optimization and Applications - 11th International Conference, COCOA 2017, Shanghai, China, December 16-18, 2017, Proceedings, Part II*, volume 10628 of *Lecture Notes in Computer Science*, pages 475–482, 2017.

2 Matthias Bentert, Till Fluschnik, André Nichterlein, and Rolf Niedermeier. Parameterized aspects of triangle enumeration. *J. Comput. Syst. Sci.*, 103:61–77, 2019. doi:10.1016/j.jcss.2019.02.004.

3 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

4 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009. doi:10.1016/j.jcss.2009.04.001.

5 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discret. Math.*, 28(1):277–305, 2014. doi:10.1137/120880240.

6 Marin Bougeret, Bart M. P. Jansen, and Ignasi Sau. Bridge-depth characterizes which structural parameterizations of vertex cover admit a polynomial kernel. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 16:1–16:19, 2020. doi:10.4230/LIPIcs.ICALP.2020.16.

7 Vasek Chvátal. Recognizing decomposable graphs. *Journal of Graph Theory*, 8(1):51–53, 1984. doi:10.1002/jgt.3190080106.

8 Bruno Courcelle. Linear delay enumeration and monadic second-order logic. *Discret. Appl. Math.*, 157(12):2675–2700, 2009. doi:10.1016/j.dam.2008.08.021.

9 Nadia Creignou, Arne Meier, Julian-Steffen Müller, Johannes Schmidt, and Heribert Vollmer. Paradigms for parameterized enumeration. *Theory Comput. Syst.*, 60(4):737–758, 2017. doi:10.1007/s00224-016-9702-4.

10 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

11 Peter Damaschke. Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. *Theor. Comput. Sci.*, 351(3):337–350, 2006. doi:10.1016/j.tcs.2005.10.004.

12 Peter Damaschke. Fixed-parameter enumerability of cluster editing and related problems. *Theory Comput. Syst.*, 46(2):261–283, 2010.

13 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

14 Henning Fernau. On parameterized enumeration. In *Computing and Combinatorics, 8th Annual International Conference, COCOON 2002, Singapore, August 15-17, 2002, Proceedings*, volume 2387 of *Lecture Notes in Computer Science*, pages 564–573. Springer, 2002. doi:10.1007/3-540-45655-4_60.

15 Fedor V. Fomin, Daniel Lokshtanove, Saket Saurabh, and Meirav Zehavi. *Kernelization. Theory of Parameterized Preprocessing*. Cambridge University Press, Cambridge, 2019. doi:10.1017/9781107415157.

16 Fedor V. Fomin, Saket Saurabh, and Yngve Villanger. A polynomial kernel for proper interval vertex deletion. *SIAM J. Discret. Math.*, 27(4):1964–1976, 2013. doi:10.1137/12089051X.

17 Robert Ganian. Twin-cover: Beyond vertex cover in parameterized algorithmics. In Dániel Marx and Peter Rossmanith, editors, *Parameterized and Exact Computation - 6th International Symposium, IPEC 2011, Saarbrücken, Germany, September 6-8, 2011. Revised Selected Papers*, volume 7112 of *Lecture Notes in Computer Science*, pages 259–271. Springer, 2011. doi:10.1007/978-3-642-28050-4_21.

18 Robert Ganian. Improving vertex cover as a graph parameter. *Discret. Math. Theor. Comput. Sci.*, 17(2):77–100, 2015. URL: http://dmtcs.episciences.org/2136.

19 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

20 Petr Golovach, Christian Komusiewicz, Dieter Kratsch, and Van Bang Le. Refined notions of parameterized enumeration kernels with applications to matching cut enumerations. *CoRR*, abs2101.03800, 2021. arXiv:2101.03800.

21 Guilherme C. M. Gomes and Ignasi Sau. Finding cuts of bounded degree: Complexity, FPT and exact algorithms, and kernelization. In Bart M. P. Jansen and Jan Arne Telle, editors, *14th International Symposium on Parameterized and Exact Computation, IPEC 2019, September 11-13, 2019, Munich, Germany*, volume 148 of *LIPIcs*, pages 19:1–19:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.IPEC.2019.19.

22 R. L. Graham. On primitive graphs and optimal vertex assignments. *Ann. New York Acad. Sci.*, 175:170–186, 1970.

23 Sun-Yuan Hsieh, Hoàng-Oanh Le, Van Bang Le, and Sheng-Lung Peng. Matching cut in graphs with large minimum degree. In Ding-Zhu Du, Zhenhua Duan, and Cong Tian, editors, *Computing and Combinatorics - 25th International Conference, COCOON 2019, Xi'an, China, July 29-31, 2019, Proceedings*, volume 11653 of *Lecture Notes in Computer Science*, pages 301–312. Springer, 2019. doi:10.1007/978-3-030-26176-4_25.

24 George Karakostas. A better approximation ratio for the vertex cover problem. *ACM Trans. Algorithms*, 5(4):41:1–41:8, 2009. doi:10.1145/1597036.1597045.

25 Christian Komusiewicz, Dieter Kratsch, and Van Bang Le. Matching cut: Kernelization, single-exponential time fpt, and exact exponential algorithms. *Discret. Appl. Math.*, 283:44–58, 2020. doi:10.1016/j.dam.2019.12.010.

26 Christian Komusiewicz and Johannes Uhlmann. A cubic-vertex kernel for flip consensus tree. *Algorithmica*, 68(1):81–108, 2014.

27 Dieter Kratsch and Van Bang Le. Algorithms solving the matching cut problem. *Theor. Comput. Sci.*, 609:328–335, 2016. doi:10.1016/j.tcs.2015.10.016.

28 Stefan Kratsch. Recent developments in kernelization: A survey. *Bull. EATCS*, 113, 2014.

29 Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy kernelization. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 224–237. ACM, 2017. doi:10.1145/3055399.3055456.

30 Andrea Marino. *Analysis and enumeration*, volume 6 of *Atlantis Studies in Computing*. Atlantis Press, Paris, 2015. Algorithms for biological graphs, With forewords by Tiziana Calamoneri and Pierluigi Crescenzi.

31 Kitty Meeks. Randomised enumeration of small witnesses using a decision oracle. *Algorithmica*, 81(2):519–540, 2019.

32 Arne Meier. *Parametrised enumeration*. Habilitation thesis, Leibniz Universit'at Hannover, 2020. doi:10.15488/9427.

33 Sang-il Oum. Approximating rank-width and clique-width quickly. *ACM Trans. Algorithms*, 5(1):10:1–10:20, 2008. doi:10.1145/1435375.1435385.

34 Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B*, 96(4):514–528, 2006. doi:10.1016/j.jctb.2005.10.006.

35 Marko Samer and Stefan Szeider. Backdoor trees. In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 363–368. AAAI Press, 2008. URL: http://www.aaai.org/Library/AAAI/2008/aaai08-057.php.

36 Marc Tedder, Derek G. Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, volume 5125 of *Lecture Notes in Computer Science*, pages 634–645. Springer, 2008. doi:10.1007/978-3-540-70575-8_52.

37 Kunihiro Wasa. Enumeration of enumeration algorithms. *CoRR*, abs/1605.05102, 2016. arXiv:1605.05102.