Towards a Theory of Parameterized Streaming Algorithms

Rajesh Chitnis
School of Computer Science, University of Birmingham, UK
rajeshchitnis@gmail.com

Graham Cormode
University of Warwick, UK
g.cormode@warwick.ac.uk

Abstract

Parameterized complexity attempts to give a more fine-grained analysis of the complexity of problems: instead of measuring the running time as a function of only the input size, we analyze the running time with respect to additional parameters. This approach has proven to be highly successful in delineating our understanding of NP-hard problems. Given this success with the TIME resource, it seems but natural to use this approach for dealing with the SPACE resource. First attempts in this direction have considered a few individual problems, with some success: Fafianie and Kratsch [MFCS'14] and Chitnis et al. [SODA'15] introduced the notions of streaming kernels and parameterized streaming algorithms respectively. For example, the latter shows how to refine the $\Omega(n^2)$ bit lower bound for finding a minimum Vertex Cover (VC) in the streaming setting by designing an algorithm for the parameterized k-VC problem which uses $O(k^2 \log n)$ bits.

In this paper, we initiate a systematic study of graph problems from the paradigm of parameterized streaming algorithms. We first define a natural hierarchy of space complexity classes of `FPS`, `SubPS`, `SemiPS`, `SupPS` and `BrutePS`, and then obtain tight classifications for several well-studied graph problems such as Longest Path, Feedback Vertex Set, Dominating Set, Girth, Treewidth, etc. into this hierarchy (see Figure 1 and Figure 2). On the algorithmic side, our parameterized streaming algorithms use techniques from the FPT world such as bidimensionality, iterative compression and bounded-depth search trees. On the hardness side, we obtain lower bounds for the parameterized streaming complexity of various problems via novel reductions from problems in communication complexity. We also show a general (unconditional) lower bound for space complexity of parameterized streaming algorithms for a large class of problems inspired by the recently developed frameworks for showing (conditional) kernelization lower bounds.

Parameterized algorithms and streaming algorithms are approaches to cope with TIME and SPACE intractability respectively. It is our hope that this work on parameterized streaming algorithms leads to two-way flow of ideas between these two previously separated areas of theoretical computer science.

2012 ACM Subject Classification Theory of Computation → Design and analysis of algorithms

Keywords and phrases Parameterized Algorithms, Streaming Algorithms, Kernels

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.7

Funding *Rajesh Chitnis*: Work done while at University of Warwick, UK and supported by ERC grant 2014-CoG 647557.
Graham Cormode: Supported by ERC grant 2014-CoG 647557.

Acknowledgements We thank MohammadTaghi Hajiaghayi, Robert Krauthgamer and Morteza Monemizadeh for helpful discussions. Algorithm 1 was suggested to us by Arnold Filtser.

1 Introduction

Designing and implementing efficient algorithms is at the heart of computer science. Traditionally, efficiency of algorithms has been measured with respect to running time as a

function of instance size. From this perspective, algorithms are said to be efficient if they can be solved in time which is bounded by some polynomial function of the input size. However, very many interesting problems are NP-complete, and so are grouped together as "not known to be efficient". This fails to discriminate within a large heterogenous group of problems, and in response the theory of *parameterized (time) algorithms* was developed in late 90's by Downey and Fellows [20]. Parameterized complexity attempts to delineate the complexity of problems by expressing the costs in terms of additional parameters. Formally, we say that a problem is *fixed-parameter tractable* (FPT) with respect to parameter k if the problem can be solved in time $f(k) \cdot n^{O(1)}$ where f is a computable function and n is the input size. For example, the problem of checking if a graph on n vertices has a vertex cover of size at most k can be solved in $2^k \cdot n^{O(1)}$ time. The study of various parameters helps to understand which parameters make the problem easier (FPT) and which ones cause it to be hard. The parameterized approach towards NP-complete problems has led to development of various algorithmic tools such as kernelization, iterative compression, color coding, and more [21, 14].
Kernelization: A key concept in fixed parameter tractability is that of kernelization which is an efficient preprocessing algorithm to produce a smaller, equivalent output called the "kernel". Formally, a kernelization algorithm for a parameterized problem Q is an algorithm which takes as an instance $\langle x, k\rangle$ and outputs in time polynomial in $(|x| + k)$ an equivalent[1] instance $\langle x', k'\rangle$ such that $\max\{|x'|, k'\} \leq f(k)$ for some computable function f. The output instance $\langle x', k'\rangle$ is called the kernel, while the function f determines the size of the kernel. Kernelizability is equivalent to fixed-parameter tractability, and designing compact kernels is an important question. In recent years, (conditional) lower bounds on kernels have emerged [3, 16, 17, 22, 27].
Streaming Algorithms: A very different paradigm for handling large problem instances arises in the form of streaming algorithms. The model is motivated by sources of data arising in communication networks and activity streams that are considered to be too big to store conveniently. This places a greater emphasis on the space complexity of algorithms. A streaming algorithm processes the input in one or a few read-only passes, with primary focus on the storage space needed. In this paper we consider streaming algorithms for graph problems over fixed vertex sets, where information about the edges arrives edge by edge [29]. We consider variants where edges can be both inserted and deleted, or only insertions are allowed. We primarily consider single pass streams, but also give some multi-pass results.

1.1 Parameterized Streaming Algorithms and Kernels

Given that parameterized algorithms have been extremely successful for the TIME resource, it seems natural to also use it attack the SPACE resource. In this paper, we advance the model of parameterized streaming algorithms, and start to flesh out a hierarchy of complexity classes. We focus our attention on graph problems, by analogy with FPT, where the majority of results have addressed graphs. From a space perspective, there is perhaps less headroom than when considering the time cost: for graphs on n vertices, the entire graph can be stored using $O(n^2)$ space[2]. Nevertheless, given that storing the full graph can be prohibitive, there are natural space complexity classes to consider. We formalize these below, but informally, the classes partition the dependence on n as: (i) (virtually) independent of n; (ii) sublinear in n; (iii) (quasi)linear in n; (iv) superlinear but subquadratic in n; and (v) quadratic in n.

[1] By equivalent we mean that $\langle x, k\rangle \in Q \Leftrightarrow \langle x', k'\rangle \in Q$
[2] Throughout the paper, by space we mean words/edges/vertices. Each word can be represented using $O(\log n)$ bits

Naively, several graph problems have strong lower bounds: for example, the problem of finding a minimum vertex cover on graphs of n vertices has a lower bound of $\Omega(n^2)$ bits. However, when we adopt the parameterized view, we seek streaming algorithms for (parameterized) graph problems whose space can be expressed as a function of *both* the number of vertices n and the parameter k. With this relaxation, we can separate out the problem space and start to populate our hierarchy. We next spell out our results, which derive from a variety of upper and lower bounds building on the streaming and FPT literature.

1.2 Our Results & Organization of the paper

For a graph problem with parameter k, there can be several possible choices for the space complexity needed to solve it in the streaming setting. In this paper, we first define some natural space complexity classes below:

1. *$\widetilde{O}(f(k))$ space:* Due to the connection to running time of `FPT` algorithms, we call the class of parameterized problems solvable using $\widetilde{O}(f(k))$ bits as `FPS` (fixed-parameterized streaming)[3].
2. *Sublinear space:* When the dependence on n is sublinear, we call the class of parameterized problems solvable using $\widetilde{O}(f(k)\cdot n^{1-\epsilon})$ bits as `SubPS` (sublinear parameterized streaming)
3. *Quasi-linear space:* Due to the connection to the semi-streaming model [26, 33], we call the set of problems solvable using $\widetilde{O}(f(k)\cdot n)$ bits as `SemiPS` (parameterized semi-streaming).
4. *Superlinear, subquadratic space:* When the dependence on n is superlinear (but subquadratic), we call the class of parameterized problems solvable using $\widetilde{O}(f(k)\cdot n^{1+\epsilon})$ bits (for some $1 > \epsilon > 0$) as `SupPS` (superlinear parameterized streaming).
5. *Quadratic space:* We call the set of graph problems solvable using $O(n^2)$ bits as `BrutePS` (brute-force parameterized streaming). Note that every graph problem is in `BrutePS` since we can just store the entire adjacency matrix using $O(n^2)$ bits (see Remark 2).

▶ Remark 1. Formally, we need t consider the following 7-tuple when we attempt to find its correct position in the aforementioned hierarchy of complexity classes:

[Problem, Parameter, Space,# of Passes, Type of Algorithm, Approx. Ratio, Type of Stream]

By type of algorithm, we mean that the algorithm could be deterministic or randomized. For the type of stream, the standard alternatives are (adversarial) insertion, (adversarial) insertion-deletion, random order, etc. Figure 3 gives a list of results for the k-VC problem (as a case study) in various different settings. Unless stated otherwise, throughout this paper, we consider the space requirement for 1-pass exact deterministic algorithms for problems with the standard parameter (size of the solution)on insertion-only streams.

▶ Remark 2. There are various different models for streaming algorithms depending on how much computation is allowed on the stored data. In this paper, we consider the most general model by allowing *unbounded computation* at each edge update, and also at the end of the stream.

Our goal is to provide a tight classification of graph problems into the aforementioned complexity classes. We make progress towards this goal as follows: Section 2 shows how various techniques from the FPT world such as iterative compression, branching, bidimensionality, etc. can also be used to design parameterized streaming algorithms. First we investigate

[3] Throughout this paper, we use the $\tilde{O}$ notation to hide $\log^{O(1)} n$ factors

whether one can further improve upon the FPS algorithm of Chitnis et al. [9] for k-VC which uses $O(k^2 \cdot \log n)$ bits and one pass. We design two algorithms for k-VC which use $O(k \cdot \log n)$ bits[4]: an 2^k-pass algorithm using bounded-depth search trees (Section 2.1) and an $(k \cdot 2^{2k})$-pass algorithm using iterative compression (Section 2.2). Finally, Section 2.3 shows that any minor-bidimensional problem belongs to the class SemiPS.

Section 3 deals with lower bounds for parameterized streaming algorithms. First, in Section 3.1 we show that some parameterized problems are tight for the classes SemiPS and BrutePS. In particular, we show that k-Treewidth, k-Path and k-Feedback-Vertex-Set are tight for the class SemiPS, i.e., they belong to SemiPS but do not belong to the sub-class SubPS. Our SemiPS algorithms are based on problem-specific structural insights. Via reductions from the Perm problem [35], we rule out algorithms which use $\widetilde{O}(f(k) \cdot n^{1-\epsilon})$ bits (for any function f and any $\epsilon \in (0,1)$) for these problems by proving $\Omega(n \log n)$ bits lower bounds for constant values of k. Then we show that some parameterized problems such as k-Girth and k-Dominating-Set are tight for the class BrutePS, i.e, they belong to BrutePS but do not belong to the sub-class SupPS. Every graph problem belongs to BrutePS since we can store the entire adjacency matrix of the graph using $O(n^2)$ bits. Via reductions from the Index problem [30], we rule out algorithms which use $\widetilde{O}(f(k) \cdot n^{1+\epsilon})$ bits (for any function f and any $\epsilon \in (0,1)$) for these problems by proving $\Omega(n^2)$ bits lower bounds for constant values of k.

Section 3.2 shows a lower bound of $\Omega(n)$ bits for any algorithm that approximates (within a factor $\frac{\beta}{32}$) the size of min dominating set on graphs of arboricity $(\beta + 2)$, i.e., this problem has no $\widetilde{O}(f(\beta) \cdot n^{1-\epsilon})$ bits algorithm (since β is a constant), and hence does not belong to the class SubPS when parameterized by β. In Section 3.3 we obtain unconditional lower bounds on the space complexity of 1-pass parameterized streaming algorithms for a large class of graph problems inspired by some of the recent frameworks to show conditional lower bounds for kernels [3, 16, 17, 22, 27]. In the full version we also show that any parameterized streaming algorithm for the d-SAT problem (for any $d \geq 2$) must (essentially) follow the naive algorithm of storing all the clauses.

Figure 1 provides a pictorial representation of the complexity classes, and the known classification of several graph problems (from this paper and some previous work) into these classes. Figure 2 summarizes our results, and clarifies the stream arrival model(s) under which they hold.

1.3 Prior work on Parametrized Streaming Algorithms

Prior work began by considering how to implement kernels in the streaming model. Formally, a streaming kernel [25] for a parameterized problem (I, k) is a streaming algorithm that receives the input I as a stream of elements, stores $f(k) \cdot \log^{O(1)} |I|$ bits and returns an equivalent instance[5]. This is especially important from the practical point of view since several real-world situations can be modeled by the streaming setting, and streaming kernels would help to efficiently preprocess these instances. Fafianie and Kratsch [25] showed that the kernels for some problems like Hitting Set and Set Matching can be implemented in the streaming setting, but other problems such as Edge Dominating Set, Feedback Vertex Set, etc. do not admit (1-pass) streaming kernels.

Chitnis et al. [9] studied how to circumvent the worst case bound of $\Omega(n^2)$ bits for Vertex

[4] Which is essentially optimal since the algorithm also returns a VC of size k (if one exists)
[5] [25] required $f(k) = k^{O(1)}$, but we choose to relax this requirement

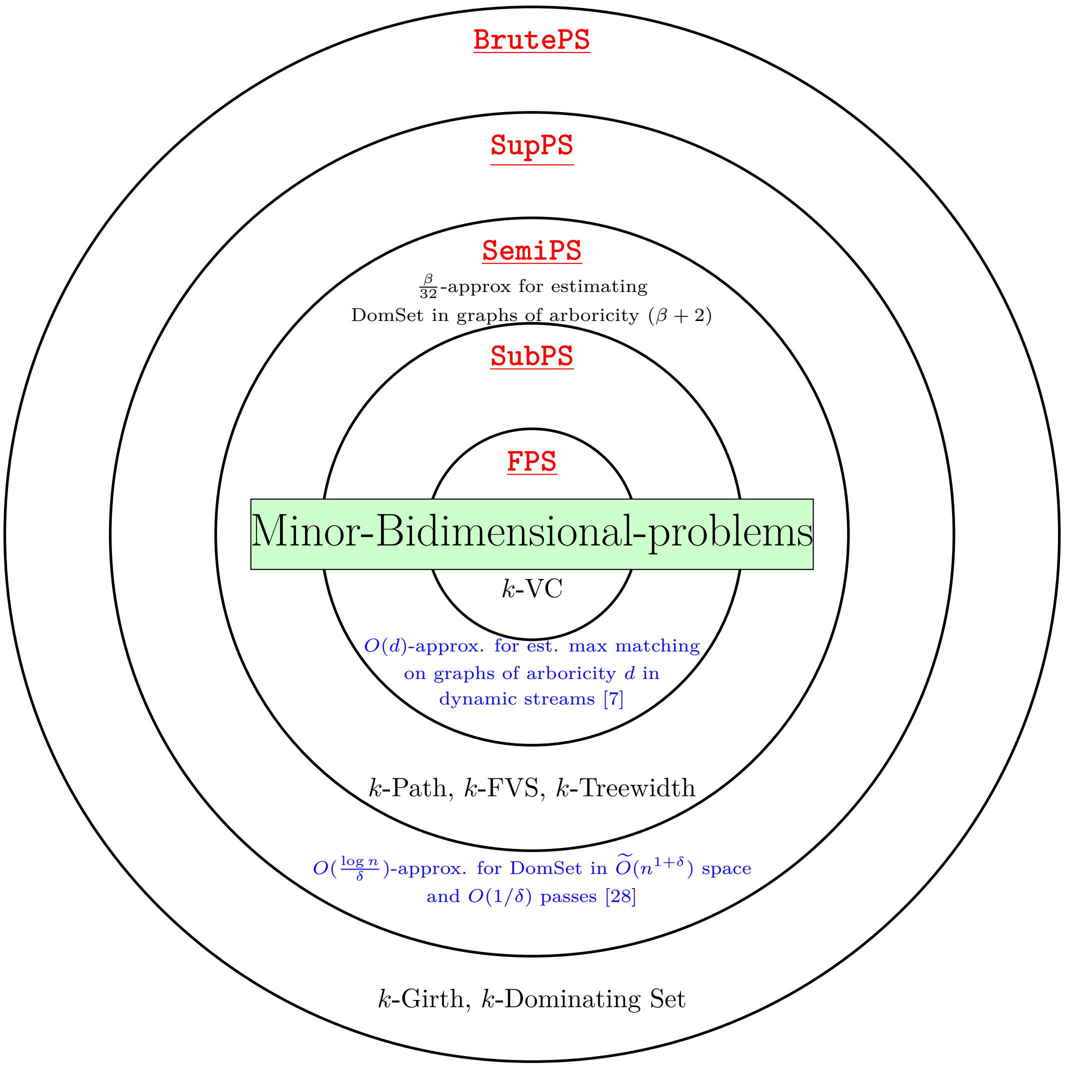
BrutePS
SupPS
SemiPS
$\frac{\beta}{32}$-approx for estimating
DomSet in graphs of arboricity $(\beta+2)$
SubPS
FPS
Minor-Bidimensional-problems
k-VC
$O(d)$-approx. for est. max matching
on graphs of arboricity d in
dynamic streams [7]
k-Path, k-FVS, k-Treewidth
$O(\frac{\log n}{\delta})$-approx. for DomSet in $\widetilde{O}(n^{1+\delta})$ space
and $O(1/\delta)$ passes [28]
k-Girth, k-Dominating Set

Figure 1 Pictorial representation of classification of some graph problems into complexity classes: our results are in black and previous work is referenced in blue. All results are for 1-pass deterministic algorithms on insertion-only streams unless otherwise specified. It was already known that k-VC $\in$ FPS [9, 7] using only 1-pass, but here we design an algorithm with optimal space storage at the expense of multiple passes.

Cover by designing a streaming algorithm for the parameterized k-Vertex-Cover (k-VC)[6]. They showed that the k-VC problem can be solved in insertion-only streams using storage of $O(k^2)$ space. They also showed an almost matching lower bound of $\Omega(k^2)$ bits for any streaming algorithm for k-VC. A sequence of papers showed how to solve the k-VC problem in more general streaming models: Chitnis et al. [9, 8] gave an $\widetilde{O}(k^2)$ space algorithm under a particular promise, which was subsequently removed in [7].

Recently, there have been several papers considering the problem of estimating the size of a maximum matching using $o(n)$ space in graphs of bounded arboricity. If the space

[6] That is, determine whether there is a vertex cover of size at most k?

Problem	Number of Passes	Type of Stream	Space Upper Bound	Space Lower Bound
$g(r)$-minor-bidimensional problems [Sec. 2.3]	1	Ins-Del.	$\widetilde{O}((g^{-1}(k+1))^{10}n)$ words	—
k-VC [Sec. 2.2]	$2^{2k} \cdot k$	Ins-only	$O(k)$ words	$\Omega(k)$ words
k-VC [Sec. 2.1]	2^k	Ins-only	$O(k)$ words	$\Omega(k)$ words
k-FVS, k-Path k-Treewidth [Sec. 3.1]	1	Ins-only	$O(k \cdot n)$ words	No $f(k) \cdot n^{1-\epsilon} \log^{O(1)} n$ bits algorithm
k-FVS, k-Path k-Treewidth [Sec. 3.1]	1	Ins-Del.	$\widetilde{O}(k \cdot n)$ words	No $f(k) \cdot n^{1-\epsilon} \log^{O(1)} n$ bits algorithm
k-Girth, k-DomSet, [Sec. 3.1]	1	Ins-Del.	$O(n^2)$ bits	No $f(k) \cdot n^{2-\epsilon} \log^{O(1)} n$ bits algorithm
$\frac{\beta}{32}$-approximation for size of min DomSet on graphs of arboricity β [Sec. 3.2]	1	Ins-only	$\widetilde{O}(n\beta)$ bits	No $f(\beta) \cdot n^{1-\epsilon}$ bits algorithm
AND-compatible problems and OR-compatible problems [Sec. 3.3]	1	Ins-only	$O(n^2)$ bits	No $\widetilde{O}(f(k) \cdot n^{1-\epsilon})$ bits algorithm
d-SAT with N variables [$\star$]	1	Clause Arrival	$\widetilde{O}(d \cdot N^d)$ bits	$\Omega((N/d)^d)$ bits

Figure 2 Table summarizing our results (in the order in which they appear in the paper). All our algorithms are deterministic. All the lower bounds are unconditional, and hold even for randomized algorithms in insertion-only streams. Proofs of results labeled with [$\star$] appear in the full version.

Problem	# of Passes	Type of Stream	Type of Algorithm	Approx. Ratio	Space Bound
k-VC	1	Ins-only	Det.	1	$O(k^2 \log n)$ bits [9]
k-VC	1	Ins-only	Rand.	1	$\Omega(k^2)$ bits [9]
k-VC	1	Ins-Del.	Rand.	1	$O(k^2 \log^{O(1)} n)$ bits [7]
k-VC	2^k	Ins-only	Det.	1	$O(k \log n)$ bits [Algorithm 1]
k-VC	$k \cdot 2^k$	Ins-only.	Det.	1	$O(k \log n)$ bits [$\star$]
Estim. k-VC	$\Omega(k/\log n)$	Ins-only.	Rand.	1	$O(k \log n)$ bits [1, Theorem 16]
Estim. k-VC on Trees	1	Ins-only.	Det. Rand.	$(3/2-\epsilon)$	$\Omega(n)$ bits [24, Theorem 6.1] $\Omega(\sqrt{n})$ bits [24, Theorem 6.1]

Figure 3 Table summarizing some of the results for the k-VC problem in the different settings outlined in Remark 1. Proofs of results labeled with [$\star$] appear in the full version.

is required to be sublinear in n, then versions of the problem that involve estimating the size of a maximum matching (rather than demonstrating such a matching) become the focus. Since the work of Esfandiari et al. [24], there have been several sublinear space algorithms [31, 32, 12, 7] which obtain $O(\alpha)$-approximate estimations of the size of maximum matching in graphs of arboricity α. The current best bounds [4, 12] for insertion-only streams is $O(\log^{O(1)} n)$ space and for insertion-deletion streams is $\widetilde{O}(\alpha \cdot n^{4/5})$. All of these results can be viewed as parameterized streaming algorithms (`FPS` or `SubPS`) for approximately estimating the size of maximum matching in graphs parameterized by the arboricity.

2 Parameterized Streaming Algorithms Inspired by FPT techniques

In this section we design parameterized streaming algorithms using three techniques from the world of parameterized algorithms, viz. branching, iterative compression and bidimensionality.

2.1 Multipass FPS algorithm for k-VC using Branching

The streaming algorithm from Section 2.2 already uses optimal storage of $O(k \log n)$ bits but requires $O(2^{2k} \cdot k)$ passes. In this section, we show how to reduce the number of passes to 2^k (while still maintaining the same storage) using the technique of bounded-depth search trees (also known as branching). The method of bounded-depth search trees gives a folklore FPT algorithm for k-VC which runs in $2^{O(k)} \cdot n^{O(1)}$ time. The idea is simple: any vertex cover must contain at least one end-point of each edge. We now build a search tree as follows: choose an arbitrary edge, say $e = u - v$ in the graph. Start with the graph G at the root node of the search tree. Branch into two options, viz. choosing either u or v into the vertex cover[7]. The resulting graphs at the two children of the root node are $G - u$ and $G - v$. Continue the branching process. Note that at each step, we branch into two options and we only need to build the search tree to height k for the k-VC problem. Hence, the binary search tree has $2^{O(k)}$ leaf nodes. If the resulting graph at any leaf node is empty (i.e., has no edges) then G has a vertex cover of size $\leq k$ which can be obtained by following the path from the root node to the leaf node in the search tree. Conversely, if the resulting graphs at none of the leaf nodes of the search tree are empty then G does not have a vertex cover of size $\leq k$: this is because at each step we branched on all the (two) possibilities at each node of the search tree.

Simulating branching-based FPT algorithm using multiple passes: We now simulate the branching-based FPT algorithm described in the previous section using 2^k passes and $O(k \log n)$ bits of storage in the streaming model.

▶ **Definition 3.** *Let $V(G) = \{v_1, v_2, \ldots, v_n\}$. Fix some ordering ϕ on $V(G)$ as follows: $v_1 < v_2 < v_3 < \ldots < v_n$. Let $\texttt{Dict}_k$ be the dictionary ordering on the 2^k binary strings of $\{0,1\}^k$. Given a string $X \subseteq \{0,1\}^k$, let $\texttt{Dict}_k(\texttt{Next}(X))$ denote the string that comes immediately after X in the ordering $\texttt{Dict}_k$. We set $\texttt{Dict}_k(\texttt{Next}(1^k)) = \spadesuit$*

We formally describe our multipass algorithm in Algorithm 1. This algorithm crucially uses the fact that in each pass we see the edges of the stream in the *same* order.

▶ **Theorem 4.** [$\star$] [8] *Algorithm 1 correctly solves the k-VC problem using 2^k passes and $O(k \log n)$ bits of storage.*

Note that the total storage of Algorithm 1 is $O(k \log n)$ bits which is essentially optimal since the algorithm also outputs a vertex cover of size at most k (if one exists).

The next natural question is whether one need exponential (in k) number of passes when we want to solve the k-VC problem using only $O(k \log n)$ bits. A lower bound of $(k/\log n)$ passes follows for such algorithms from the following result of Abboud et al.

▶ **Theorem 5.** (rewording of [1, Thm 16]) *Any algorithm for the k-VC problem which uses S bits of space and R passes must satisfy $RS \geq n^2$*

[7] Note that if we choose u in the first branch then that does not imply that we cannot or will not choose v later on in the search tree

[8] Proofs of results labeled with [$\star$] appear in the full version.

Algorithm 1 2^k-pass Streaming Algorithm for k-VC using $O(k \log n)$ bits via Branching

Input: An undirected graph $G = (V, E)$ and an integer k.
Output: A vertex cover S of G of size $\leq k$ (if one exists), and NO otherwise
Storage: i, j, S, X

```
1: Let $X = 0^k$, and suppose the edges of the graph are seen in the order $e_1, e_2, \ldots, e_m$
2: while $X \neq ♠$ do
       $S = \emptyset, i = 1, j = 1$
3:     while $i \neq k + 1$ do
4:         Let $e_j = u - v$ such that $u < v$ under the ordering $\phi$
5:         if Both $u \notin S$ and $v \notin S$ then
6:             if $X[i] = 0$ then $S \leftarrow S \cup \{u\}$
7:             else $S \leftarrow S \cup \{v\}$
8:             $i \leftarrow i + 1$
9:         $j \leftarrow j + 1$
10:    if $j = m + 1$ then Return $S$ and abort
11:    else $X \leftarrow \texttt{Dict}_k(\texttt{Next}(X))$
12: if $X = ♠$ then Return NO
```

2.2 Multipass FPS algorithm for k-VC using Iterative Compression

The technique of *iterative compression* was introduced by Reed et al. [34] to design the first FPT algorithm for the k-OCT problem[9]. Since then, iterative compression has been an important tool in the design of faster parameterized algorithms [6, 10, 5] and kernels [15]. In the full version, using the technique of iterative compression, we design an algorithm for k-VC which uses $O(k \log n)$ bits but requires $O(k \cdot 2^{2k})$ passes. Although this algorithm is strictly worse (same storage, but higher number of passes) compared to Algorithm 1, we mention it here to illustrate that the technique of iterative compression can be used in the streaming setting.

As in the FPT setting, a natural problem to attack using iterative compression in the streaming setting would be the k-OCT problem. It is known that 0-OCT, i..e, checking if a given graph is bipartite, in the 1-pass model has an upper bound of $O(n \log n)$ bits [26] and a lower bound of $\Omega(n \log n)$ bits [35]. For $k \geq 1$, can we design a $g(k)$-pass algorithm for k-OCT which uses $\widetilde{O}(f(k) \cdot n)$ bits for some functions f and g, maybe using iterative compression? To the best of our knowledge, such an algorithm is not known even for 1-OCT.

2.3 Minor-Bidimensional problems belong to SemiPS

The theory of bidimensionality [18, 19] provides a general technique for designing (subexponential) FPT for NP-hard graph problems on various graph classes. In this section, we briefly sketch how we can use this technique to show that a large class of problems belong to the class SemiPS. All the details (including graph-theoretic definitions such as minors, treewidth, etc.) of this section are deferred to the full version.

▶ **Definition 6 (minor-bidimensional).** *A graph problem Π is $g(r)$-minor-bidimensional if*

- *The value of Π on the $r \times r$ grid is $\geq g(r)$*

[9] Is there a set of size at most k whose deletion makes the graph odd cycle free, i.e. bipartite

- Π *is closed under taking minors, i.e., the value of* Π *does not increase under the operations of vertex deletions, edge deletions, edge contractions.*

Hence, we obtain the following "win-win" approach for designing FPT algorithms for bidimensional problems:

- Either the graph has small treewidth and we can then use dynamic programming algorithms for bounded treewidth graphs; or
- The treewidth is large[10] which implies that the graph contains a large grid as a minor. This implies that the solution size is large, since the parameter is minor-bidimensional.

Several natural graph parameters are known to be minor-bidimensional. For example, treewidth is $\Omega(r)$-minor-dimensional and Feedback Vertex Set, Vertex Cover, Minimum Maximal Matching, Long Path, etc are $\Omega(r^2)$-minor-bidimensional. To design parameterized streaming algorithms, we will replace the dynamic programming step for bounded treewidth graphs by simply storing all the edges of such graphs. The main theorem of this section is that minor-bidimensional problems belong to the class `SemiPS`.

▶ **Theorem 7.** [$\star$] ***(minor-bidimensional problems*** $\in$ ***`SemiPS`)*** *Let* Π *be a* $g(r)$*-minor-dimensional problem. Then the* k*-*Π *problem on graphs with* n *vertices can be solved using*

- $O((g^{-1}(k+1))^{10} \cdot n)$ *space in insertion-only streams*
- $\widetilde{O}((g^{-1}(k+1))^{10} \cdot n)$ *space in insertion-deletion streams*

Theorem 7 implies the following results for specific graph problems[11]:

- Since Treewidth is $\Omega(r)$-minor-bidimensional, it follows that k-Treewidth has an $O(k^{10} \cdot n)$ space algorithm in insertion-only streams and $\widetilde{O}(k^{10} \cdot n)$ space algorithm in insertion-deletion streams.
- Since problems such as Long Path, Vertex Cover, Feedback Vertex Set, Minimum Maximal Matching, etc. are $\Omega(r^2)$-minor-bidimensional, it follows that their parameterized versions have $O(k^5 \cdot n)$ space algorithm in insertion-only streams and $\widetilde{O}(k^5 \cdot n)$ space algorithm in insertion-deletion streams.

In Section 3.1, we design algorithms for some of the aforementioned problems with smaller storage. In particular, we design problem-specific structural lemmas to reduce the dependency of k on the storage from $k^{O(1)}$ to k.

▶ Remark 8. It is tempting to conjecture a lower bound complementing Theorem 7: for example, can we show that the bounds for minor-bidimensional problems are tight for `SemiPS`, i.e., they do not belong to `SubPS` or even `FPS`? Unfortunately, we can rule out such a converse to Theorem 7 via the two examples of Vertex Cover (VC) and Feedback Vertex Set (FVS) which are both $\Omega(r^2)$-minor-bidimensional. Chitnis et al. [9] showed that k-VC can be solved in $O(k^2)$ space and hence belongs to the class `FPS`. However, we show in the full version that k-FVS cannot belong to `SubPS` since it has a $\Omega(n \log n)$ bits lower bound for $k = 0$.

3 Lower Bounds for Parameterized Streaming Algorithms

3.1 Tight Problems for the classes `SemiPS` and `BrutePS`

In this section we show that certain problems are tight for the classes `SemiPS` and `BrutePS`. All of the results hold for 1-pass in the insertion-only model. Our algorithms are deterministic, while the lower bounds also hold for randomized algorithms.

[10] Chuzhoy and Tan [11] showed that treewidth $= O(r^9 \cdot \log^{O(1)} r) \Rightarrow$ there is a $r \times r$ grid minor

[11] We omit the simple proofs of why these problems satisfy the conditions of Definition 6

Tight Problems for the class SemiPS: We now show that some parameterized problems are tight for the class `SemiPS`, i.e.,

- They belong to `SemiPS`, i.e., can be solved using $\widetilde{O}(g(k) \cdot n)$ bits for some function g.
- They do not belong to `SubPS`, i.e., there is no algorithm which uses $\widetilde{O}(f(k) \cdot n^{1-\epsilon})$ bits for any function f and any constant $1 > \epsilon > 0$. We do this by showing $\Omega(n \cdot \log n)$ bits lower bounds for these problems for constant values of k.

For each of the problems considered in this section, a lower bound of $\Omega(n)$ bits (for constant values of k) was shown by Chitnis et al. [7]. To obtain the improved lower bound of $\Omega(n \cdot \log n)$ bits for constant k, we will reduce from the Perm problem defined by Sun and Woodruff [35].

> Perm
> *Input*: Alice has a permutation $\delta : [N] \to [N]$ which is represented as a bit string B_δ of length $N \log N$ by concatenating the images of $1, 2, \ldots, N$ under δ. Bob has an index $I \in [N \log N]$.
> *Goal*: Bob wants to find the I-th bit of B_δ

Sun and Woodruff [35] showed that the one-way (randomized) communication complexity of Perm is $\Omega(N \cdot \log N)$. Using the Perm problem, we show $\Omega(n \cdot \log n)$ bit lower bounds for constant values of k for various problem such as k-Path, k-Treewidth, k-Feedback-Vertex-Set, etc. We also show a matching upper bound for these problems: for each k, these problems can be solved using $O(kn \cdot \log n)$ words in insertion-only streams and $\widetilde{O}(kn \cdot \log n)$ words in insertion-deletion streams. The proofs of these results are deferred to the full version. To the best of our knowledge, the only problems known previously to be tight for `SemiPS` were k-vertex-connectivity and k-edge-connectivity [13, 35, 23].

Tight Problems for the class BrutePS: We now show that some parameterized problems are tight for the class `BrutePS`, i.e.,

- They belong to `BrutePS`, i.e., can be solved using $O(n^2)$ bits. Indeed any graph problem can be solved by storing the entire adjacency matrix which requires $O(n^2)$ bits.
- They do not belong to `SubPS`, i.e., there is no algorithm which uses $\widetilde{O}(f(k) \cdot n^{1+\epsilon})$ bits for any function f and any $\epsilon \in (0, 1)$. We do this by showing $\Omega(n^2)$ bits lower bounds for these problems for constant values of k via reductions from the Index problem.

> **Index**
> *Input*: Alice has a string $B = b_1 b_2 \ldots b_N \in \{0, 1\}^N$. Bob has an index $I \in [N]$
> *Goal*: Bob wants to find the value b_I

There is a $\Omega(N)$ lower bound on the (randomized) one-way communication complexity of Index [30]. Via reduction from the Index problem, we are able to show $\Omega(n^2)$ bits for constant values of k for several problems such as k-Dominating-Set and k-Girth. The proofs of these reductions are deferred to the full version.

▶ Remark 9. We usually only design `FPT` algorithms for `NP`-hard problems. However, parameterized streaming algorithms make sense for all graph problems since we are only comparing ourselves against the naive choice of storing all the $O(n^2)$ bits of adjacency matrix. Hence, here we consider the k-Girth problem as an example of a polynomial time solvable problem.

Finally, in the full version, we also show that for any $d \geq 2$, any streaming algorithm for d-SAT (in the clause arrival model) must essentially store all the clauses (and hence fits into the "brute-force" streaming setting). This is the only non-graph-theoretic result in this paper, and may be viewed as a "streaming analogue" of the Exponential Time Hypothesis.

3.2 Lower bound for approximating size of minimum Dominating Set on graphs of bounded arboricity

▶ **Theorem 10.** *Let $\beta \geq 1$ be any constant. Then any algorithm which $\frac{\beta}{32}$-approximates the size of a min dominating set on graphs of arboricity $\beta + 2$ requires $\Omega(n)$ space.*

Note that Theorem 10 shows that the naive algorithm which stores all the $O(n\beta)$ edges of an β-arboriticy graph is essentially optimal. Our lower bound holds even for randomized algorithms (required to have success probability $\geq 3/4$) and also under the vertex arrival model, i.e., we see at once all edges incident on a vertex. We (very) closely follow the outline from [2, Theorem 4] who used this approach for showing that any α-approximation for estimating size of a minimum dominating set in general graphs requires $\tilde{\Omega}(\frac{n^2}{\alpha^2})$ space. Because we are restricted to bounded arboriticy graphs, we cannot just sue their reduction as a black-box but need to adapt it carefully for our purposes.

Let $V(G) = [n + 1]$, and $\mathcal{F}_\beta$ be the collection of all subsets of $[n]$ with cardinality β. Consider the following distribution $\mathcal{D}_{\mathsf{est}}$ for $\mathsf{DomSet}_{\mathsf{est}}$.

Distribution $\mathcal{D}_{\mathsf{est}}$: A hard input distribution for $\mathsf{DomSet}_{\mathsf{est}}$.

- **Alice.** The input of Alice is a collection of n sets $\mathcal{S}' = \{S'_1, S'_2, \ldots, S'_n\}$ where for each $i \in [n]$ we have that $S'_i = \{i\} \cup S_i$ with S_i being a set chosen independently and uniformly at random from $\mathcal{F}_\beta$.
- **Bob.** Pick $\theta \in \{0, 1\}$ and $i^* \in [n]$ independently and uniformly at random; the input of Bob is a single set T defined as follows.
  - If $\theta = 0$, then $\overline{T} = [n] \setminus T$ is a set of size $\beta/8$ chosen uniformly at random from S_{i^*}.
  - If $\theta = 1$, then $\overline{T} = [n] \setminus T$ is a set of size $\beta/8$ chosen uniformly at random from $[n] \setminus S_{i^*}$.

Recall that $\mathrm{OPT}(\mathcal{S}', T)$ denotes the size of the minimum *dominating set* of the graph G whose edge set is given by $N[i] = \{i\} \cup S_i$ for each $i \in [n]$ and $N[n+1] = \{n+1\} \cup T$. It is easy to see that G has arboricity $\leq (\beta+2)$ since it has $(n+1)$ vertices and $\leq (\beta+1)n + (1+n-\frac{\beta}{8})$ edges. We first establish the following lemma regarding the parameter θ and $\mathrm{OPT}(\mathcal{S}', T)$ in the distribution $\mathcal{D}_{\mathsf{est}}$.

▶ **Lemma 11.** [$\star$] *Let $\alpha = \frac{\beta}{32}$. Then, for $(\mathcal{S}', T) \sim \mathcal{D}_{\mathsf{est}}$ we have*
1. $\Pr\left(OPT(\mathcal{S}', T) = 2 \mid \theta = 0\right) = 1$.
2. $\Pr\left(OPT(\mathcal{S}', T) > 2\alpha \mid \theta = 1\right) = 1 - o(1)$.

The proof of Lemma 11 is deferred to the full version. The first observation is that the distribution $\mathcal{D}_{\mathsf{est}}$ is not a product distribution due to the correlation between the input given to Alice and Bob. However, we can express the distribution $\mathcal{D}_{\mathsf{est}}$ as a convex combination of a relatively small set of product distributions. The proof of Theorem 10 then follows by showing a lower bound on this set of product distributions. This proof is a bit technical, and we defer it to the full version.

3.3 Streaming Lower Bounds Inspired by Kernelization Lower Bounds

Streaming algorithms and kernelization are two (somehwhat related) compression models. In kernelization, we have access to the whole input but our computation power is limited to polynomial time whereas in streaming algorithms we don't have access to the whole graph (have to pay for whatever we store) but have unbounded computation power on whatever part of the input we have stored.

A folklore result states that a (decidable) problem is FPT if and only if it has a kernel. Once the fixed-parameter tractability for a problem is established, the next natural goals are to reduce the running time of the FPT algorithm and reduce the size of the kernel. In the last decade, several frameworks have been developed to show (conditional) lower bounds on the size of kernels [3, 16, 17, 22, 27]. Inspired by these frameworks, we define a class of problems, which we call as AND-compatible and OR-compatible, and show (unconditionally) that none of these problems belong to the class `SubPS`.

▶ **Definition 12.** *We say that a graph problem* Π *is* AND-compatible *if there exists a constant* k *such that*

- *for every* $n \in \mathbb{N}$ *there exists a graph* G_{YES} *of size* n *such* $\Pi(G_{\text{YES}}, k)$ *is a YES instance*
- *for every* $n \in \mathbb{N}$ *there exists a graph* G_{NO} *of size* n *such* $\Pi(G_{\text{NO}}, k)$ *is a NO instance*
- *for every* $t \in \mathbb{N}$ *we have that* $\Pi\Big(\uplus_{i=1}^{t} G_i, k\Big) = \wedge_{i=1}^{t}\Pi(G_i, k)$ *where* $G = \uplus_{i=1}^{t} G_i$ *denotes the union of the vertex-disjoint graphs* $G_1, G_2, \ldots, G_t$

Examples of *AND-compatible* graph problems are k-Treewidth, k-Girth, k-Pathwidth, k-Coloring, etc.

▶ **Definition 13.** *We say that a graph problem* Π *is* OR-compatible *if there exists a constant* k *such that*

- *for every* $n \in \mathbb{N}$ *there exists a graph* G_{YES} *of size* n *such* $\Pi(G_{YES}, k)$ *is a YES instance*
- *for every* $n \in \mathbb{N}$ *there exists a graph* G_{NO} *of size* n *such* $\Pi(G_{NO}, k)$ *is a NO instance*
- *for every* $t \in \mathbb{N}$ *we have that* $\Pi(\uplus_{i=1}^{t} G_i, k) = \vee_{i=1}^{t}\Pi(G_i, k)$ *where* $G = \uplus_{i=1}^{t} G_i$ *denotes the union of the vertex-disjoint graphs* $G_1, G_2, \ldots, G_t$

A general example of an *OR-compatible* graph problem is the subgraph isomorphism problem parameterized by size of smaller graph: given a graph G of size n and a smaller graph H of size k, does G have a subgraph isomorphic to H? Special cases of this problem are k-Path, k-Clique, k-Cycle, etc.

▶ **Theorem 14.** *If* Π *is an* AND-compatible *or an* OR-compatible *graph problem then* $\Pi \notin$ *SubPS*

Proof. Let Π be an *AND-compatible* graph problem, and $G = \uplus_{i=1}^{t} G_i$ for some $t \in \mathbb{N}$. We claim that any streaming algorithm ALG for Π must use t bits. Intuitively, we need at least one bit to check that each of the instances (G_i, k) is a YES instance of Π (for all $1 \leq i \leq t$). Consider a set of t graphs $\mathcal{G} = \{G_1, G_2, \ldots, G_t\}$: note that we don't fix any of these graphs yet. For every subset $X \subseteq [t]$ we define the instance (G_X, k) of Π where $G_X = \uplus_{j\in J} G_j$. Suppose that ALG uses less than t bits. Then by pigeonhole principle, there are two subsets I, I' of $[t]$ such that ALG has the same answer on (G_I, k) and $(G_{I'}, k)$. Since $I \neq I'$ (without loss of generality) there exists i^* such that $i^* \in I \setminus I'$. This is where we now fix each of the graphs in $\mathcal{G}$ to arrive at a contradiction: consider the input where $G_i = G_{\text{YES}}$ for all $(I \cup I') \setminus i^*$ and $G_{i^*} = G_{\text{NO}}$. Then, it follows that (G_I, k) is a NO instance but $(G_{I'}, k)$ is a YES instance.

Suppose that $\Pi \in$ `SubPS`, i.e., there is an algorithm for Π which uses $f(k) \cdot N^{1-\epsilon} \cdot \log^{O(1)} N$ bits (for some $1 > \epsilon > 0$) on a graph G of size N to decide whether (G, k) is a YES or NO instance. Let $G = \uplus_{i=1}^{t} G_i$ where $|G_i| = n$ for each $i \in [t]$. Then $|G| = N = nt$. By the previous paragraph, we have that

$$f(k) \cdot (nt)^{1-\epsilon} \cdot \log^{O(1)}(nt) \geq t \Rightarrow f(k) \cdot n^{1-\epsilon} \cdot \log^{O(1)}(nt) \geq t^{\epsilon}$$

Choosing $t = n^{\frac{2-\epsilon}{\epsilon}}$ we have that $f(k) \cdot \log^{O(1)} n^{1+(\frac{2-\epsilon}{\epsilon})} \geq n$, which is a contradiction for large enough n (since k and ϵ are constants).

We now prove the lower bound for AND-compatible problems. Recall that De Morgan's law states that $\neg(\vee_i P_i) = \wedge_i(\neg P_i)$. Hence, if Π is an *OR-compatible* graph problem then the complement[12] problem $\overline{\Pi}$ is an *AND-compatible* graph problem, and hence the lower bound follows from the previous paragraph. ◀

▶ Remark 15. Note that throughout this paper we have considered the model where we allow unbounded computation at each edge update, and also at the end of the stream. However, if we consider a **restricted** model of allowing only polynomial (in input size n) computation at each edge update and also at end of the stream, then it is easy to see that existing (conditional) lower bounds from the parameterized algorithms and kernelization setting translate easily to this restricted model. For example, the following two lower bounds for parameterized streaming algorithms follow immediately in the restricted (polytime computation) model:

- Let X be a graph problem that is $W[i]$-hard parameterized by k (for some $i \geq 1$). Then (in the polytime computation model) $X \notin \texttt{FPS}$ unless $\texttt{FPT} = W[i]$.
- Let X be a graph problem that is known to not have a polynomial kernel unless $\texttt{NP} \subseteq \texttt{coNP}/\text{poly}$. Then (in the polytime computation model) X does not have a parameterized streaming algorithm which uses $k^{O(1)} \cdot \log^{O(1)} n$ bits, unless $\texttt{NP} \subseteq \texttt{coNP}/\text{poly}$.

4 Conclusions & Open Problems

In this paper, we initiate a systematic study of graph problems from the paradigm of parameterized streaming algorithms. We define space complexity classes of `FPS`, `SubPS`, `SemiPS`, `SupPS` and `BrutePS`, and then obtain tight classifications for several well-studied graph problems such as Longest Path, Feedback Vertex Set, Girth, Treewidth, etc. into these classes. Our parameterized streaming algorithms use techniques of bidimensionality, iterative compression and branching from the FPT world. In addition to showing lower bounds for some parameterized streaming problems via communication complexity, we also show how (conditional) lower bounds for kernels and W-hard problems translate to lower bounds for parameterized streaming algorithms.

Our work leaves open several concrete questions. We list some of them below:

- The streaming algorithm (Algorithm 1) for k-VC (on insertion-only streams) from Section 2.1 has an optimal storage of $O(k \log n)$ bits but requires 2^k passes. Can we reduce the number of passes to poly(k), or instead show that we need passes which are superpolynomial in k if we restrict space usage to $O(k \log n)$ bits? The only known lower bound for such algorithms is $(k/\log n)$ passes (see Theorem 5).
- For $k \geq 1$ can we design algorithms which use $f(k) \cdot n \cdot \log^{O(1)} n$ bits and $g(k)$ passes for the k-OCT problem (for some functions f, g)? The technique of iterative compression seems like a natural tool to use here.

References

1 Amir Abboud, Keren Censor-Hillel, Seri Khoury, and Ami Paz. Smaller Cuts, Higher Lower Bounds. *CoRR*, abs/1901.01630, 2019. URL: `http://arxiv.org/abs/1901.01630`, `arXiv:1901.01630`.

[12] By complement, we mean that $\overline{\Pi}(G, k)$ is YES if and only if $\Pi(G, k)$ is NO

2 Sepehr Assadi, Sanjeev Khanna, and Yang Li. Tight bounds for single-pass streaming complexity of the set cover problem. In *STOC*, pages 698–711, 2016.

3 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.

4 Marc Bury, Elena Grigorescu, Andrew McGregor, Morteza Monemizadeh, Chris Schwiegelshohn, Sofya Vorotnikova, and Samson Zhou. Structural Results on Matching Estimation with Applications to Streaming. *Algorithmica*, 81(1):367–392, 2019.

5 Jianer Chen, Fedor V. Fomin, Yang Liu, Songjian Lu, and Yngve Villanger. Improved algorithms for feedback vertex set problems. *J. Comput. Syst. Sci.*, 74(7):1188–1198, 2008.

6 Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5):21:1–21:19, 2008.

7 Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In *SODA*, pages 1326–1344, 2016.

8 Rajesh Hemant Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, and Morteza Monemizadeh. Brief announcement: New streaming algorithms for parameterized maximal matching & beyond. In *SPAA*, pages 56–58, 2015.

9 Rajesh Hemant Chitnis, Graham Cormode, Mohammad Taghi Hajiaghayi, and Morteza Monemizadeh. Parameterized streaming: Maximal matching and vertex cover. In *SODA*, pages 1234–1251, 2015.

10 Rajesh Hemant Chitnis, Marek Cygan, Mohammad Taghi Hajiaghayi, and Dániel Marx. Directed Subset Feedback Vertex Set Is Fixed-Parameter Tractable. *ACM Trans. Algorithms*, 11(4):28:1–28:28, 2015.

11 Julia Chuzhoy and Zihan Tan. Towards Tight(er) Bounds for the Excluded Grid Theorem. In *SODA*, pages 1445–1464, 2019.

12 Graham Cormode, Hossein Jowhari, Morteza Monemizadeh, and S. Muthukrishnan. The Sparse Awakens: Streaming Algorithms for Matching Size Estimation in Sparse Graphs. In *ESA*, pages 29:1–29:15, 2017.

13 Michael S. Crouch, Andrew McGregor, and Daniel Stubbs. Dynamic Graphs in the Sliding-Window Model. In *ESA*, pages 337–348, 2013.

14 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

15 Frank K. H. A. Dehne, Michael R. Fellows, Frances A. Rosamond, and Peter Shaw. Greedy Localization, Iterative Compression, Modeled Crown Reductions: New FPT Techniques, an Improved Algorithm for Set Splitting, and a Novel $2k$ Kernelization for Vertex Cover. In *IWPEC*, pages 271–280, 2004.

16 Holger Dell. AND-Compression of NP-Complete Problems: Streamlined Proof and Minor Observations. *Algorithmica*, 75(2):403–423, 2016.

17 Holger Dell and Dieter van Melkebeek. Satisfiability Allows no Nontrivial Sparsification Unless the Polynomial-Time Hierarchy Collapses. In *STOC*, pages 251–260, 2010.

18 Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. ACM*, 52(6):866–893, 2005.

19 Erik D. Demaine and MohammadTaghi Hajiaghayi. The bidimensionality theory and its algorithmic applications. *Comput. J.*, 51(3):292–302, 2008.

20 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity.* Monographs in Computer Science. Springer, 1999.

21 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, 2013.

22 Andrew Drucker. New Limits to Classical and Quantum Instance Compression. In *FOCS*, pages 609–618, 2012.

23 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997.

24 Hossein Esfandiari, Mohammad Taghi Hajiaghayi, Vahid Liaghat, Morteza Monemizadeh, and Krzysztof Onak. Streaming Algorithms for Estimating the Matching Size in Planar Graphs and Beyond. In *SODA*, pages 1217–1233, 2015.

25 Stefan Fafianie and Stefan Kratsch. Streaming kernelization. In *MFCS*, pages 275–286, 2014.

26 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005.

27 Lance Fortnow and Rahul Santhanam. Infeasibility of Instance Compression and Succinct PCPs for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011.

28 Sariel Har-Peled, Piotr Indyk, Sepideh Mahabadi, and Ali Vakilian. Towards Tight Bounds for the Streaming Set Cover Problem. In *PODS*, pages 371–383, 2016.

29 Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. *External memory algorithms*, 50:107–118, 1998.

30 Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1997.

31 Andrew McGregor and Sofya Vorotnikova. Planar Matching in Streams Revisited. In *APPROX/RANDOM*, pages 17:1–17:12, 2016.

32 Andrew McGregor and Sofya Vorotnikova. A Simple, Space-Efficient, Streaming Algorithm for Matchings in Low Arboricity Graphs. In *SOSA*, pages 14:1–14:4, 2018.

33 S. Muthukrishnan. Data Streams: Algorithms and Applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

34 Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004.

35 Xiaoming Sun and David P. Woodruff. Tight Bounds for Graph Problems in Insertion Streams. In *APPROX-RANDOM*, pages 435–448, 2015.