Exponential Time Paradigms Through the Polynomial Time Lens*

Andrew Drucker[1], Jesper Nederlof[†2], and Rahul Santhanam[‡3]

1 **Computer Science Department, University of Chicago, USA.**
andy.drucker@gmail.com
2 **Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands.** J.Nederlof@tue.nl
3 **Department of Computer Science, University of Oxford, United Kingdom.**
rahul.santhanam@cs.ox.ac.uk

Abstract

We propose a general approach to modelling algorithmic paradigms for the exact solution of NP-hard problems. Our approach is based on polynomial time reductions to succinct versions of problems solvable in polynomial time. We use this viewpoint to explore and compare the power of paradigms such as branching and dynamic programming, and to shed light on the true complexity of various problems.

As one instantiation, we model branching using the notion of witness compression, i.e., reducibility to the circuit satisfiability problem parameterized by the number of variables of the circuit. We show this is equivalent to the previously studied notion of 'OPP-algorithms', and provide a technique for proving conditional lower bounds for witness compressions via a constructive variant of AND-composition, which is a notion previously studied in theory of preprocessing. In the context of parameterized complexity we use this to show that problems such as Pathwidth and Treewidth and Independent Set parameterized by pathwidth do not have witness compression, assuming $NP \not\subseteq coNP/poly$. Since these problems admit fast fixed parameter tractable algorithms via dynamic programming, this shows that dynamic programming can be stronger than branching, under a standard complexity hypothesis. Our approach has applications outside parameterized complexity as well: for example, we show if a polynomial time algorithm outputs a maximum independent set of a given planar graph on n vertices with probability $\exp(-n^{1-\epsilon})$ for some $\epsilon > 0$, then $NP \subseteq coNP/poly$. This negative result dims the prospects for one very natural approach to sub-exponential time algorithms for problems on planar graphs.

As two other illustrations (more exploratory) of our approach, we model algorithms based on inclusion-exclusion or group algebras via the notion of "parity compression", and we model a subclass of dynamic programming algorithms with the notion of "disjunctive dynamic programming". These models give us a way to naturally classify various parameterized problems with FPT algorithms. In the case of the dynamic programming model, we show that Independent Set parameterized by pathwidth is complete for this model.

1998 ACM Subject Classification F.2.0. Analysis of Algorithms and Problem Complexity

Keywords and phrases exponential time paradigms, branching, dynamic programming, lower bounds

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.[183]

* This work was partly done while the authors were visiting the Simons Institute for the Theory of Computing during the program 'Fine-Grained Complexity and Algorithm Design' in the fall of 2015.
† Funded by the NWO VENI project 639.021.438.
‡ Supported by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013)/ERC Grant Agreement no. 615075


© Andrew Drucker, Jesper Nederlof and Rahul Santhanam;
licensed under Creative Commons License CC-BY

1 Introduction

The successes of theoretical computer science have often been driven by simple but general algorithmic approaches, or *paradigms*, leading to efficient algorithms in many different application domains. Indeed, paradigms such as divide-and-conquer, branching, dynamic programming and linear programming have been applied over and over to design algorithms.

A natural question that arises is to *quantify* the power and limitations of a given algorithmic paradigm. Doing so may have several benefits. It can help us understand what makes the paradigm effective. It can make algorithm design and analysis less ad hoc, with greater clarity about when and for which problems the paradigm is relevant. It can also enable us to compare various algorithmic paradigms with each other in terms of their power and usefulness. A crucial challenge in studying the power of algorithmic paradigms is the *modelling* question. We need a modelling framework which is rich enough to capture existing, successful algorithms within the paradigm. On the other hand, we need the modelling framework to be meaningfully restricted, so that we can prove interesting things about these models and the limits of their power. These goals are often in tension.

We aim to model exponential time algorithms. Understanding what can be computed in exponential time seems to be harder than understanding what can be computed in polynomial time, and less is known. In particular, showing general exponential-time lower bounds based on standard hypotheses about *polynomial-time* computation (for example, the hypotheses that $P \neq NP$ or that the Polynomial Hierarchy is infinite) seems out of reach. We propose to bypass this issue by arguing that several specific, established algorithmic paradigms can be modelled as polynomial-time reductions to (succinct) problems, so that limitations to their power may follow from these kinds of traditional hypotheses.

Approaches to algorithmic modelling can be broadly classified into syntactic and semantic approaches. Syntactic approaches attempt to faithfully represent the step-by-step operation of algorithms conforming to the method. Examples include the modelling of **1.** DPLL algorithms by proof systems such as Resolution, **2.** backtracking and dynamic programming by certain kinds of branching programs [1], **3.** dynamic programming by feasible dominance relations [22], **4.** linear programming by extended formulations [9]. These approaches, though natural, suffer from some drawbacks. The first is their lack of flexibility—they can fail to capture simple-looking variants of the method, e.g., the failure of proof systems to capture randomization. Second, in the search for accuracy, the models produced by such approaches can get quite complicated, which makes them hard to analyze.

Our models, in contrast, are semantic—we try to capture broad features of the algorithmic method rather than trying to model it in a step-by-step fashion. In particular, we allow arbitrary polynomial-time computations as constituent subroutines. This allows the model to flexibly accommodate preprocessing and natural variants of the method, and makes sense for the intended applications to exponential time algorithms. Our use of parameterization enables us to distinguish between algorithmic methods in a way that a traditional complexity-theoretic approach cannot. Although our approach is coarser than most syntactic approaches, it is more uniform, applying to a variety of algorithmic methods at once, and enables us to get useful information about the relative power of these methods.

Related Previous Work

A large number of problems have been shown to be *Fixed Parameter Tractable (FPT)*, i.e., solvable in time $O^*(f(k))$, where k is a parameter provided with each input, $O^*(\cdot)$ suppresses factors polynomial in the input size, and $f(\cdot)$ is some computable function. For

many problems we now know essentially the optimal running time: there is an $O^*(f(k))$ time algorithm and an $O^*(g(k))$ time algorithm for any $g(k) < f(o(k))$ contradicts the Exponential Time Hypothesis (ETH). For a few problems we even know that $O^*(f(k))$ time algorithms cannot be improved to $O^*(f(k)^{1-\Omega(1)})$ time algorithm under stronger hypotheses as the Strong ETH. In this work we are mostly interested in problems for which $f(k) = 2^{\mathrm{poly}(k)}$ - this is the case for most natural FPT parameterizations of NP-complete problems.

Kernelization. A natural paradigm to prove a problem is solvable in $O^*(2^{\mathrm{poly}(k)})$ time is preprocessing plus brute force: given an instance (x, k) of a parameterized problem, transform it in polynomial time to an instance (x', k') of the same problem where $|x'|, k'$ are polynomial in k (this part is called the *polynomial kernel*), and then solve the smaller instance using brute-force search.[1] The power of polynomial kernelization has been extensively investigated, and is by now fairly well understood. For many parameterized problems, we have either found a polynomial kernel, or showed they do not exist unless $NP \subseteq coNP/poly$; the latter is proved by providing an *(OR* or *AND)-composition*, and appealing to results in [4, 19, 16] and related works. This fits as an excellent starting point for our study since it gives a lower bound for a class of exponential-time algorithms modelled via polynomial-time reductions, and is conditional on an hypothesis concerning polynomial-time computation.

Branching. Another heavily used paradigm to solve a problem in $O^*(2^{\mathrm{poly}(k)})$ time is that of *branching*, or *bounded search trees*. A natural model for this paradigm is the model of *One-sided Probabilistic Polynomial (OPP) algorithms* proposed by Paturi and Pudlak [33] in their study of algorithms for satisfiability. OPP algorithms are polynomial-time algorithms with one-sided error which never accept no-instances but only detect yes-instances with small but non-trivial probability (called the *success probability*). An OPP algorithm with success probability $f(n)$ can be converted to a bounded-error randomized algorithm running in time $\mathrm{poly}(n)/f(n)$ just by taking the OR of $f(n)$ independent trials. On the other hand if an exponential-time algorithm can be thought of as traversing an exponential-size recursion tree which performs polynomial-time checks at leaves and returns true if at some leaf true is returned, then we can cast this as an OPP algorithm provided we are able to sample leaves of the branching tree in an efficient, nearly uniform way (in [33], this observation was attributed to Eppstein [17]). We would like to remark that OPP is more powerful than one might think at first sight as it also directly captures, for example, Schöning's algorithm [36].

Concerning lower bounds, Paturi and Pudlak [33] showed that OPP algorithms with success probability significantly better than 2^{-n} for circuit satisfiability on n variables would have unlikely consequences. Particularly relevant for our work is work by Drucker [15] showing a $2^{-n^{1-\epsilon}}$ upper bound of OPP algorithms' success probability for 3-CNF-SAT (for any $\epsilon > 0$), assuming $NP \not\subseteq coNP/poly$.

Several closely-related formalisms of branching algorithms have been proposed in the literature [6, 35, 40]. In the context of parameterized complexity, Marx proposed a study of branching [29, 30] using a model 'BFPT' of branching FPT algorithms.[2] Also relevant is work of Dantsin and Hirsch [13], which discusses a notion closely related to our notion of witness compression in the context of exact algorithms for Satisfiability, and provides lower bounds conditioned on ETH.

[1] That is, try all bit-strings and see if a certificate arises.
[2] That turns out to be equivalent to OPP algorithms with success probability $2^{-O(k)}$.

Our Contribution

In this work, we argue that many contemporary exponential-time algorithms can be rewritten as polynomial-time reductions to succinct version of problems in P, and we also give several concrete results on the applicability of specific algorithmic paradigms to different problems. We outline these results next.

Branching. Our main technical contributions address branching algorithms as modelled by OPP algorithms or equivalently witness compressions (defined below). Building on machinery developed by Drucker [15] we give lower bounds for constructive OPP algorithms. For instance:

▶ **Theorem 1.1.** *If there is a polynomial time algorithm that, given a planar graph on n vertices, outputs a maximum independent set with probability $\exp(-O(n^{1-\epsilon}))$ for some $\epsilon > 0$, then $NP \subseteq coNP/poly$.*

Note that $\exp(O(\sqrt{n}))$ time algorithms are known (e.g. [26]), so this indicates that a rich class of branching algorithms is incapable of exploiting planarity for solving independent set. We also give a simple OPP algorithm that actually establishes success probability $\exp(-O(n/\sqrt{\log(n)}))$.

Following a hashing lemma from [33], we observe that having an OPP algorithm with success probability $f(k)$ is equivalent to having a polynomial-time Monte Carlo reduction from the problem at hand to CKT-SAT[3] with $1/\log(f(k))$ input gates. Thus in the generic context sketched in this paper, the succinct problem corresponding to our model of branching is CKT-SAT. If $f(k) = 2^{-\mathrm{poly}(k)}$, there are witnesses for the problem of size $\mathrm{poly}(k)$ and we will refer to the polynomial time Monte Carlo reduction as a *polynomial witness compression* since a satisfying solution of the circuit that the reduction outputs can be seen as a witness for the original instance to be a yes-instance. We call a witness compression *Levin* or *constructive* if we can determine a solution of the original problem given a satisfying assignment of the circuit.

We define a type of reduction we call 'constructive AND-composition' that is closely related to AND-compositions from kernelization theory, and show that assuming $NP \not\subseteq coNP/poly$ no parameterized problem can both have a constructive AND-composition and a Levin polynomial witness compression. As one particular application, we use this to *separate* dynamic programming from branching (as modelled via OPP algorithms). Specifically, we show that Independent Set parameterized by pathwidth,[4] which is known to be FPT via a dynamic programming algorithm, does not have Levin polynomial witness compressions unless $NP \not\subseteq coNP/poly$. An important question[5] is how fast this problem can be solved using only polynomial space. In [27], the authors provide an $O^*(2^{O(\mathtt{pw}^2)})$-time and polynomial-space algorithm based on a tradeoff between dynamic programming and Savitch's theorem, but the folklore dynamic programming algorithm uses $O^*(2^{\mathtt{pw}})$ time and space (see also Section E). Our results thus indicate that branching algorithms of the OPP type, a very natural class of polynomial space algorithms, will not be useful here.

We emphasize that the model of OPP algorithms and witness compressions are powerful by observing that problems such as Steiner Tree, Long Path and Directed Feed-

[3] Refer to Section 2 for a definition.
[4] That is, we assume a path decomposition of width $\mathtt{pw}$ is given as input.
[5] This question first appeared in print in [27], but was explicitly asked before at least in [31].

back Vertex Set (DFVS) do have polynomial witness compressions as a consequence of methods from previous works.

Kernelization. The above results on branching have a number of consequences for kernelization theory. To explain these, let us first stress that it seems that if a problem has an AND-composition it seems very likely it also has an constructive AND-composition since all known AND-compositions are known to be constructive.

There has been interest recently in relaxed versions of kernelization, such as OR-kernels, where rather than computing one small instance from the initial instance, we compute a list of instances, at least one of which is in the language if and only if the original instance was. It is easy to see that a polynomial witness compression is a far reaching generalization of OR-kernelization: if a problem has a OR-kernel the witness would indicate which output of the OR-kernel is a yes-instance along with a certificate of this instance being a YES. On the other hand, a problem as CKT-SAT with k input variables is known to not have polynomial kernelization assuming $NP \not\subseteq coNP/poly$ (see e.g., [14]) but trivially has a polynomial witness compression. Our observation thus implies that problems cannot have both constructive AND-compositions and OR-kernelizations simultaneously unless $NP \subseteq coNP/poly$.

Our connection between constructive AND-composition and witness compressions combined with the polynomial witness compressions for Steiner Tree, Long Path and DFVS implies that these problems do not have constructive AND-compositions, which is a clear indication that they do not admit AND-compositions as studied in kernelization theory. We feel this is a useful insight especially for DFVS because the existence of a polynomial compression for this is a major open problem [10], and since we currently only know how to exclude polynomial compressions via AND- and OR-compressions our connection indicates we probably should not look for AND-compressions.

Parity Compression. There are several other important paradigms that in many cases seem essential to known algorithms for various problems, especially to obtain the best known bounds on the function $f(k)$. In [33], the authors mention as examples the paradigms of exponential-time divide-and-conquer; inclusion-exclusion; dynamic programming; group algebra; and Voronoi cell decomposition; and they argue that 'OPP and its generalizations could serve as an excellent starting point for the study of exponential-time algorithms for NP-complete problems in general', although they leave such generalizations unspecified.

We further explore this direction, using our unifying perspective via succinct parameterized problems. Similar to witness compression, we define a notion of "parity compression" corresponding to reducibility to the problem $\oplus$CKT-SAT parameterized by the number of variables. The idea here is that algebraic and inclusion-exclusion based approaches to FPT algorithms often implicitly reduce the problem to a succinctly represented parity of exponentially many input bits, i.e, an instance of $\oplus$CKT-SAT. We illustrate this phenomenon by capturing the Long Path and K-Cycle problems in our model.

Disjunctive Dynamic Programming. We model a subclass of dynamic programming algorithms which we refer to as "disjunctive dynamic programming". Intuitively, this corresponds to dynamic programming tables whose entries are Boolean ORs of lexicographically-prior entries. We model this class via reducibility to the problem CNF-Reach, an instance of which is a directed graph succinctly encoded by a CNF, with the question being whether there is a source-sink path of a prescribed length in the graph. The parameter is the number

of variables of the CNF. Essentially, the existence of a path corresponds to a "trace" of a disjunctive dynamic programming algorithm with a YES answer.

More generally, one could study succinctness implemented by circuits rather than CNFs; however, the choice of CNFs has a nice benefit: it allows us to find natural complete problems for our model. Specifically, we show that Independent Set parameterized by pathwidth is complete for this model, thus in some sense dynamic programming is the "right" algorithmic technique for this problem. The completeness of Independent Set parameterized by pathwidth may also be interpreted as another signal that polynomial space algorithms for problems parameterized by pathwidth might be hard to find as they need to exploit the succinctness given by the CNF representation or otherwise need to improve over Savitch's theorem for short reachability. Let us remark that related research has been proposed earlier: the reduction as outlined here has been conjectured in the second author's PhD-thesis [32], and the aforementioned signal was remarked in [32, 2, 34]

Organization This work is organized as follows: in Section 2 we provide a few preliminaries. Note that due to space constraints we do not cover basic definitions from parameterized complexity such as definitions of fixed-parameter tractability that provide context for our work; we refer the reader to a recent textbook [11]. Section 3 presents our main technical results, which are on branching algorithms. Section 4 introduces the model of parity compression, Section 5 introduces the model of disjunctive dynamic programming and in Section 6 we list a number of interesting directions for further research.

2 Preliminaries and Notation

For an integer p, $[p] := \{1, \ldots, p\}$, and $\binom{X}{p}$ denotes the family of size-p subsets of a set X.

Probabilistic Circuits. A *probabilistic circuit* is a (De Morgan) Boolean circuit $C(x, r)$ which, in addition to its input gates $x \in \{0,1\}^n$, has a designated set of "randomness gates" $r \in \{0,1\}^{\mathrm{poly}(n)}$. We say such a circuit computes a function $f(x)$ with success probability $p(n)$ if, for all $x \in \{0,1\}^n$, $\Pr_r[C(x,r) = f(x)] \geq p(n)$. Here the probability is taken over a uniform random setting to r. By Cook's transformation, any polynomial time randomized algorithm can be expressed as a (logspace-uniform) family of polynomial-size probabilistic circuits.

Problem Definitions. PC denotes the set of search problems whose solutions can be verified in polynomial time (following [20]). For $L \subseteq \{0,1\}^*$, χ_L denotes the characteristic vector of L. A *parameterized problem* is a set $Q \subseteq \{0,1\}^* \times \mathbb{N}$.

We use the following notation to define (parameterized) (search) problems in NP or PC: if k is some parameter of an unparameterized problem R, R/k denotes the associated problem parameterized by k. When a problem has a natural search version, we will use this to define it, as the decision version follows from the search version. We use L_Q to denote the decision version of a search problem Q. The following parameterized search problems will be important for this paper:

CKT-Sat **Parameter:** n.

Instance: A Boolean circuit C on n variables.

Witness: An assignment $x \in \{0,1\}^n$ such that $C(x) = 1$.

(d)-CNF-Sat **Parameter:** n.

Instance: A Boolean (d)-CNF-formula C on n variables.

Witness: An assignment $x \in \{0,1\}^n$ such that $C(x) = 1$.

For any search problem $R \in PC$, we define a search problem $AND(R)$ as follows:

$AND(R)$ **Parameter:** n.

Instance: Instances $x_1, \ldots, x_t \in \{0,1\}^n$

Witness: $y_1, \ldots, y_t$ such that $(x_i, y_i) \in R$ for every i.

Reductions. For search problems[6] $Q, R \in PC$, a *Levin reduction* from Q to R consists of two polynomial time algorithms, A_1 and A_2, such that (i) $\exists y : (x,y) \in Q$ if and only if $\exists y' : (A_1(x), y') \in R$, (ii) if $(A_1(x), y') \in R$, then $(x, A_2(x,y')) \in Q$. A *Monte Carlo reduction* from language L to language L' is a randomized polynomial time algorithm that takes $x \in \{0,1\}^*$ as input and outputs $y \in \{0,1\}^*$ such that (i) if $x \notin L$ then $y \notin L'$, (ii) if $x \in L$ then $\Pr[y \in L'] \geq 1/4$. A *Levin Monte Carlo reduction* from search problem Q to search problem R is a pair of two randomized polynomial time algorithms A and B with the following properties: (i) A is a Monte Carlo reduction from L_Q to L_R mapping x to x', (ii) B takes as input x, x' and y', and if $(x', y') \in R$, then with probability $1/4$, B outputs y such that $(x,y) \in Q$.

Success Probability of Polynomial Time Algorithms. Let $f : \mathbb{N} \times \mathbb{N} \to \mathbb{R}$. We say that an algorithm *solves a parameterized problem Q with success probability f*, if given (x,k) it returns NO if $(x,k) \notin L_Q$ and YES with probability at least $f(|x|,k)$ if $(x,k) \in L_Q$. Moreover, if $Q \in PC$, *it finds solutions for Q* with probability at least f if given (x,k) it returns NO if $(x,k) \notin L_Q$ and it returns a certificate for $(x,k) \in L_Q$ with probability at least $f(|x|,k)$, otherwise. Note that an algorithm finding solutions for Q also solves Q.

By standard boosting arguments we see that if there is a polynomial time algorithm solving Q or finding solutions for Q with probability at least f, then for any polynomial p there is also a polynomial time algorithm solving Q or finding solutions for Q with probability at least $\min\{\frac{1}{2}, p(|x|)f(|x|,k)\}$. Therefore, if $f(|x|,k)$ is $1/(\text{poly}(|x|)f(k))$, we say it solves or finds solutions for Q with probability at least $f'(k)$ where $f'(k) = f(1,k)$.

Non-deterministic Direct Product Reductions. For a function $f : A \to B$ and integer t, we denote $f^{\otimes t} : A^t \to B^t$ to be the t-fold direct product of f, e.g., for $x_1, \ldots, x_t \in A$ we let $f^{\otimes t}(x_1, \ldots, x_t) = (f(x_1), \ldots, f(x_t))$. The following result will be crucial for this work:

▶ **Theorem 2.1** (Theorem 1.2 of [15]). *Let $f = \{f_N\}$ be a family of Boolean functions on N input bits, and suppose that $f \notin NP/poly \cap coNP/poly$. Let $100 \leq t(N) \leq \text{poly}(N)$ be a parameter and let $\{C_N\}_{N>0}$ be any family of polynomial-size probabilistic circuits outputting $t(N)$ bits. Then for infinitely many choices of N and $x \in \{0,1\}^{N \times t(N)}$,*

$$\Pr[C_N = f_N^{\otimes t(N)}(x)] < \exp(-\Omega(t(N)). \tag{2.1}$$

3 Branching via OPP Algorithms and Witness Compressions

In this section we present our results on branching algorithms. We first formally define the notion of constructive AND-compositions and state how they exclude OPP algorithms.

[6] In this work, we implicitly cast a parameterized (search) problem as a normal (search) problem by omitting the parameter where convenient.

Then we formally introduce witness compressions and show their close relation with OPP algorithms. Subsequently, we point out implications to parameterized complexity.

Constructive AND-Compositions and Their Consequences.

▶ **Definition 3.1** (Constructive AND-composition)**.** Let L be a search problem, Q be a parameterized search problem and d be a constant. We say that a pair of algorithms (A,B) is a *constructive AND-composition of degree d from L into Q* if the following conditions hold:

1. A is given $x_1, x_2, \ldots, x_t$ and outputs an instance $(x, k) \in \Sigma^* \times \mathbb{N}$ in time polynomial in $\sum_{i=1}^{t} |x_i|$ such that $k \leq \text{poly}(\max_i |x_i| \log(t))$ and $|x| \leq \text{poly}(\max_i |x_i| \log(t)) t^d$,
2. if for every i there exist y_i such that $(x_i, y_i) \in L$, then B does the following: B takes as input $x_1, x_2, \ldots, x_t$, the instance (x, k), and a certificate y such that $(x, k, y) \in Q$, and outputs y_i for every i such that

$$\Pr[\forall i : (x_i, y_i) \in L] \geq \exp\left(-\text{poly}\left(\max_i |x_i|\right) \log(t)\right).$$

This is closely related to AND-compositions as studied in kernelization complexity (see e.g. [11, Section 15.1.3]): it is more strict in the sense that the reduction needs to be Levin, but more general in the sense that we only need a weak probabilistic guarantee on the output. We will see that even constructive AND-compositions of degree 1 with trivial parameterizations have interesting consequences.

▶ **Theorem 3.2.** *If there is a constructive AND-composition of degree d from a PC-hard search problem L into a parameterized search problem Q, then no polynomial time algorithm finds solutions for every instance (x, k) of Q with probability $\exp(-\text{poly}(k)|x|^{1/d-\Omega(1)})$, unless $NP \subseteq coNP/poly$.*

As one concrete application we obtain the Theorem as mentioned in the introduction:

▶ **Theorem 1.1** (restated)**.** If there is a polynomial time algorithm that, given a planar graph on n vertices, outputs a maximum independent set with probability $\exp(-O(n^{1-\epsilon}))$ for some $\epsilon > 0$, then $NP \subseteq coNP/poly$.

Proof. Let L be the following search problem: given the adjacency list of a planar graph G and integer θ, find an independent set of G of size at least θ. The decision variant of this problem NP-complete and by inspecting the known reductions, the problem is also seen to be PC-complete. Let Q be L with a trivial parameterization (e.g., the parameter equals 1). We now give a constructive AND-composition of degree 1 from L to Q. Given instances $(G_1 = (V_1, E_1), \theta_1), \ldots, (G_t = (V_t, E_t), \theta_t)$, create an instance (G, θ^*) of Q where G is the disjoint union of $G_1, \ldots, G_t$ (i.e. it has each graph G_i as a connected component in it), and θ^* is picked uniformly at random from $\{1, \ldots, \sum_{i=1}^{t} |V_i|\}$. We see that with probability $1/\sum_{i=1}^{t} |V_i| \geq \exp\left(-\text{poly}\left(\max_i |x_i|\right) \log(t)\right)$, we have that θ^* equals the size of the maximum independent set. Moreover, if we are given a maximum independent set of G, its intersection with every component must be a maximum independent set in that component so if all instances are YES instances we find maximum independent sets of size at least θ_i in G_i for every i. Since (G, θ) is represented with $\text{poly}(\max_i |V_i|) t \log(t)$ bits, we therefore found a constructive AND-composition of degree 1, and no polynomial time algorithm finds solutions for Q with probability $\exp(-|x|^{1-\Omega(1)})$ by Theorem 3.2. This implies the statement since $|x|$ is $n \log n$ for n-vertex graphs. ◀

We remark the naïve guessing procedure here is not optimal (the proof is in Appendix G):

▶ **Theorem 3.3.** *There exists a polynomial time algorithm that outputs a maximum independent set of a planar graph on n vertices with probability* $\exp(-n/\sqrt{\log n})$.

Witness Compressions. We will now give an equivalent interpretation of OPP algorithms that paves the way for defining models of other paradigms in the next sections.

▶ **Definition 3.4** ((Levin) Witness Compression)**.** A *(Levin) $h(k, N)$-witness compression* for a parameterized (search) problem Q is a (Levin) Monte-Carlo reduction from Q to CKT-SAT that maps (x, k) with $|x| = N$ to (y, n) with $n \leq h(k, N)$.

Note that having a $h(k, N)$-witness compression is equivalent to having a $h(k, N+\lg(N))$-witness compression since we can brute-force over all assignments of $\lg(N)$ input bits in polynomial time. We say a (Levin) $h(k, N)$-witness compression is *polynomial* if $h(k, N) \leq \text{poly}(k)$ (or equivalently $\text{poly}(k) + \log(N)$). The following lemma shows the equivalence of witness compression and OPP algorithms.

▶ **Lemma 3.5.** *A parameterized (search) problem has a* (Levin) $h(k, N)$-witness compression *if and only if there is a polynomial time algorithm solving it (respectively, finding solutions) with success probability at least* $2^{-h(k,N)}$.

The proof is postponed to Appendix A. The forward direction in both variants is immediate. For the backward direction, we use the 'Hash-Down lemma' from [33] to prove both variants. Our proof of the equivalence takes advantage of the fact that we allow randomized reductions in the definition of witness compression. If we were to only allow deterministic reductions in the definition, the equivalence would still hold under a sufficiently strong derandomization hypothesis - we omit the details.

We emphasize the power of polynomial witness compression by revisiting a few FPT-algorithms and observing that they give rise to efficient witness compressions. Marx [30] observes that VERTEX COVER and FEEDBACK VERTEX SET have a witness compression with $h(k)$ linear. Here, we add a few non-trivial witness compressions to this list with $h(k)$ quasi-linear. The relevant problem statements and proof of the following theorem can be found in Appendix B. All these results go via the connection from Theorem 3.5.

▶ **Theorem 3.6.** STEINER TREE *and* LONG PATH *have Levin $O(k \log k)$-witness compressions, and* DFVS *has a Levin $O(k \log^3(k))$-witness compression.*

Implications to Parameterized Complexity. As mentioned in the introduction, polynomial witness compression appears significantly more powerful than polynomial kernelization. Indeed if a problem has an OR-kernelization,[7] it is easily seen we have a polynomial witness compression:

▶ **Observation 3.7.** If Q admits a polynomial (Levin) OR-kernelization to a problem in NP, it admits a polynomial (Levin) witness compression.

On the other hand, let us remark here that there may be problems in NP that admit polynomial compressions[8] but no polynomial witness compression: Wahlström [39] gives

[7] Where rather than computing one small instance from the initial instance as in kernelization, we compute a list of instances at least one of which is in the language if and only if the original instance was, see e.g.[25] where the name 'disjunctive kernelization' was used.

[8] A compression is a kernelization where the target problem might be different (and crucially here, not even in NP).

an interesting polynomial compression of the K-cycle problem (see Appendix F for the problem definition) to a language that is not in NP, and remarks that this seems to separate polynomial kernelization from polynomial compression since it is not clear whether K-cycle has polynomial witness compressions.

The above connection is relevant for kernelization complexity because Theorem 3.8 suggests that parameterized problems with AND-compositions have no OR-kernelizations. Another interesting consequence obtained by combining Theorem 3.2 and Theorem 3.6 is (since the problems at hand are easily seen to be PC-complete):

▶ **Theorem 3.8.** Steiner Tree, Long Path, *and* DFVS *do not admit constructive AND-compositions unless* $NP \subseteq coNP/poly$.

As mentioned before, this is a useful fact especially for DFVS because the existence of a polynomial compressions for this is a big open problem [10], and we currently only know how to exclude polynomial compressions via AND- and OR-compressions Theorem 3.8. So this indicates researchers attacking this open problem probably should not look for AND-compressions.

Another useful implication concerns the following parameterized problem:

Independent Set (IS/pw) **Parameter:** pw

Instance: A graph G, path decomposition of G of width pw, integer θ.

Witness: An independent set of G of size at least θ.

As mentioned in the introduction, it is an important open question how fast this problem can be solved using only polynomial space. We show that branching algorithms (which is a subset of all polynomial space algorithms) are not useful here:

▶ **Theorem 3.9.** *Suppose a polynomial time algorithm takes as input a path decomposition of width* pw *of a graph G on n vertices, and outputs with probability* $\exp(-\text{poly}(\texttt{pw})n^{1-\Omega(1)})$ *a maximum independent set of G, then $NP \subseteq coNP/poly$.*

Since the proof is very similar to the proof of Theorem 1.1, it is postponed to Appendix H. Let us remark that several other interesting graph problems admit constructive AND-compositions. For example, following Lemma 7 from [4] we have that

▶ **Observation 3.10.** Let L be a parameterized graph search problem such that for any pair of graphs G_1 and G_2, and integer $k \in \mathbb{N}$, $(G_1, k) \in L \wedge (G_2, k) \in L \leftrightarrow (G_1 \cup G_2, k)$ where $G_1 \cup G_2$ is the disjoint union of G_1 and G_2. Then L admits a constructive AND-composition of degree 1.

Similar as in [4], this implies hardness for several problems. We refer to [4] for the definitions of these problems since our only goal is to point out the applicability of our framework.

▶ **Theorem 3.11.** *No polynomial time algorithm finds solutions for any instance (x, k) of* Cutwidth, Modified Cutwidth, Pathwidth, Branchwidth, Search Number, Gate Matrix Layout, *and* Front Size *with probability* $\exp(-\text{poly}(k)|x|^{1-\Omega(1)})$ *unless $NP \not\subseteq coNP/poly$.*

Proof. Following [4], we have by Observation 3.10 that all the above problems admit constructive AND-compositions. By inspection it can be seen that the reductions from these problems to CKT-Sat (which exist since all the above problems are NP-complete) are all Levin reductions, thus all problems are PC-complete. The claim follows from Theorem 3.2. ◀

4 Parity Compression

As mentioned before, witness compression tightly captures a large part of contemporary FPT-algorithms, but still far from all of them. Motivated by this, we propose the following natural generalization of witness compression, based on the definition of witness compression as a reduction to CKT-Sat. A *parity compression* is a polynomial time Monte Carlo reduction from the problem at hand to the $\oplus$CKT-Sat problem, defined as follows:

$\oplus$CKT-Sat **Parameter:** n

Instance: A Boolean circuit C on n variables

Asked: Whether the parity of the size of the set $\{x \in \{0,1\}^n : C(x) = 1\}$ is odd.

Analogous to witness compressions, we can interpret parity compressions as exponential time algorithms by solving the resulting $\oplus$CKT-Sat instance in time $2^n|x|^{O(1)}$ (the analogue of witness compressions was to solve the CKT-Sat instance by simple brute-force enumeration). By an easy application of the Isolation Lemma of [38], there is a polynomial time Monte Carlo reduction from CKT-Sat to $\oplus$CKT-Sat that increases the number of input variables by $O(\text{polylog}(|C|))$. Thus parity compression is a generalization of witness compression. No polynomial-time reduction in the reverse direction is known, and such a reduction (even randomized) would imply a collapse of PH in light of Toda's theorem [37].

While we are not yet able to show lower bounds for parity compressions since its study is still in its infancy, we do argue in Appendix D that several interesting contemporary algorithms (mainly, ones using inclusion/exclusion or group algebra) are exponential parities. This motivates a very interesting future research direction:

▶ **Open Problem 1.** Find non-trivial evidence against a polynomial time Monte Carlo reduction from CKT-Sat on n-variable circuits to $\oplus$CKT-Sat on n' circuits where $n' << n$.

Another goal would be to further exclude more polynomial space paradigms that are able to solve Independent Set parameterized by the pathwidth:

▶ **Open Problem 2.** Find evidence against a polynomial time Monte Carlo reduction from IS/pw to $\oplus$CKT-Sat where $n = o(\mathtt{pw}^2)$.

5 Disjunctive Dynamic Programming

One natural other algorithmic paradigm unaddressed so far (as highlighted in Theorem 3.9 and Open Problem 2) is dynamic programming. We focus in this work on a subclass of dynamic programming algorithms which we call 'disjunctive dynamic programming' - this corresponds to dynamic programming tables where the entries are Boolean ORs of previous entries. Specifically, we say a disjunctive dynamic programming algorithm is a polynomial time parameter reduction to the following problem:

CNF-Reach **Parameter:** n

Instance: A CNF-formula $\varphi : \{0,1\}^n \to \{0,1\}$ with m clauses and n even, integer $\ell = \text{poly}(n)$.

Witness: $x^1, \ldots, x^\ell \in \{0,1\}^{n/2}$ with $x^1 = 0 \cdots 0, x^\ell = 1 \cdots 1$ and $\varphi(x^i x^{i+1}) = 1$ for every $0 \leq i \leq \ell - 1$.

In Appendix E.1 we show that IS/PW is almost equivalent to CNF-REACH[9] by giving almost tight reductions between the two problems. Thus IS/PW can be seen as complete for the class of problems efficiently solvable with disjunctive dynamic programming. We feel such a reduction expresses the hardness of a problem typically solved with dynamic programming better than e.g., a reduction to CNF-SAT or even CKT-SAT since these problem do have small witnesses and polynomial-space algorithms. Next to Theorem 3.9, this may be seen as additional evidence that finding fast space-efficient algorithms for IS/PW might be very hard (e.g., we either need to exploit the succinct representation via CNF-formula's or find new algorithms for the directed reachability problem).

We also show that an algorithm for SET COVER is a disjunctive dynamic programming algorithm: we reduce SET COVER to CNF-REACH in Appendix E.1.2.

6 Directions for Further Research

We conclude this paper with a few open questions. First, for several problems, the existence of polynomial witness compression is open (see Appendix F for missing problem definitions):

▶ **Open Problem 3.** Do SUBSET SUM, KNAPSACK, KNAPSACK/WEIGHT-VALUE, K-CYCLE or DISJOINT PATHS have polynomial witness compressions?

Note that currently, it is not clear whether there exists a parameterized problem that has a polynomial compression but no polynomial witness compression, although as suggested in [39] the K-CYCLE would be a good candidate for such a problem.

One algorithmic paradigm not addressed is exponential time divide and conquer [21], which is also closely related to applications of Savitch's Theorem as used by [27]:

▶ **Open Problem 4.** Is there a good model of exponential time divide and conquer based on reductions to a succinct version of a natural problem? Can it solve IS/PW in $O^*(2^{o(\mathtt{pw}^2)})$ time?

Ambitiously, having finer-grained lower bounds would be very insightful:

▶ **Open Problem 5.** Can we rule out linear witness compressions for some problems with quasilinear witness compressions under standard assumptions?

References

1 Michael Alekhnovich, Allan Borodin, Joshua Buresh-Oppenheim, Russell Impagliazzo, Avner Magen, and Toniann Pitassi. Toward a model for backtracking and dynamic programming. In *20th Annual IEEE Conference on Computational Complexity (CCC 2005), 11-15 June 2005, San Jose, CA, USA*, pages 308–322. IEEE Computer Society, 2005.

2 Eric Allender, Shiteng Chen, Tiancheng Lou, Periklis A. Papakonstantinou, and Bangsheng Tang. Width-parametrized SAT: time–space tradeoffs. *Theory of Computing*, 10:297–339, 2014.

3 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

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.

[9] Such a reduction was already conjectured in the PhD-thesis of the second author [32, Page 35]

5 Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial Optimization on Graphs of Bounded Treewidth. *The Computer Journal*, 51(3):255–269, 2008.

6 Liming Cai and Jianer Chen. On the amount of nondeterminism and the power of verifying. *SIAM Journal on Computing*, 26(3):733–750, 1997.

7 Arthur Cayley. A theorem on trees. In *The Collected Mathematical Papers*, volume 13, pages 26–28. Cambridge University Press, 2009. Cambridge Books Online.

8 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), 2008.

9 Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. Extended formulations in combinatorial optimization. *4OR*, 8(1):1–48, 2010.

10 Marek Cygan, Fedor Fomin, Bart M.P. Jansen, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, and Saket Saurabh Michal Pilipczuk. Open problems for fpt school 2014.

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

12 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *FOCS*, pages 150–159, 2011.

13 Evgeny Dantsin and Edward A. Hirsch. Satisfiability certificates verifiable in subexponential time. In Karem A. Sakallah and Laurent Simon, editors, *Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings*, volume 6695 of *Lecture Notes in Computer Science*, pages 19–32. Springer, 2011.

14 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *J. ACM*, 61(4):23:1–23:27, 2014.

15 Andrew Drucker. Nondeterministic direct product reductions and the success probability of SAT solvers. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 736–745. IEEE Computer Society, 2013.

16 Andrew Drucker. New limits to classical and quantum instance compression. *SIAM J. Comput.*, 44(5):1443–1479, 2015.

17 David Eppstein. Quasiconvex analysis of multivariate recurrence equations for backtracking algorithms. *ACM Trans. Algorithms*, 2(4):492–509, October 2006.

18 Guy Even, Joseph Naor, Baruch Schieber, and Madhu Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.

19 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011.

20 Oded Goldreich. *Computational complexity - a conceptual perspective*. Cambridge University Press, 2008.

21 Yuri Gurevich and Saharon Shelah. Expected computation time for hamiltonian path problem. *SIAM J. Comput.*, 16(3):486–502, 1987.

22 Paul Helman. A common schema for dynamic programming and branch and bound algorithms. *Journal of the ACM*, 36(1):97–128, 1989.

23 Illya V. Hicks, Arie M. C. A. Koster, and Elif Kolotoğlu. Branch and tree decomposition techniques for discrete optimization. *Tutorials in Operations Research 2005*, pages 1–19, 2005.

24 Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.

25 Stefan Kratsch. Recent developments in kernelization: A survey. *Bulletin of the EATCS*, 113, 2014.

26 Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980.

27 Daniel Lokshtanov, Matthias Mnich, and Saket Saurabh. Planar k-path in subexponential time and polynomial space. In Petr Kolman and Jan Kratochvíl, editors, *Graph-Theoretic Concepts in Computer Science - 37th International Workshop, WG 2011, Teplá Monastery, Czech Republic, June 21-24, 2011. Revised Papers*, volume 6986 of *Lecture Notes in Computer Science*, pages 262–270. Springer, 2011.

28 Daniel Lokshtanov, M. S. Ramanujan, and Saket Saurabh. Linear time parameterized algorithms for subset feedback vertex set. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 935–946. Springer, 2015.

29 Dániel Marx. What's next? reductions other than kernelization. Talk at WorKer 2010: Workshop on Kernelization, Leiden, The Netherlands, 2010.

30 Dániel Marx. What's next? future directions in parameterized complexity. In Hans L. Bodlaender, Rod Downey, Fedor V. Fomin, and Dániel Marx, editors, *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370 of *Lecture Notes in Computer Science*, pages 469–496. Springer, 2012.

31 Jesper Nederlof. Saving space by algebraization. Talks at seminars in Utrecht (January 8) and UiB (Februari 11), UiB ICT Research school (May 18), STOC, Dagstuhl 'Exact Complexity of NP-hard Problems', 2010.

32 Jesper Nederlof. *Space and Time Efficient Structural Improvements of Dynamic Programming Algorithms*. PhD thesis, University of Bergen, 2011. "`http://www.win.tue.nl/~jnederlo/PhDThesis.pdf`".

33 Ramamohan Paturi and Pavel Pudlák. On the complexity of circuit satisfiability. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 241–250. ACM, 2010.

34 Michal Pilipczuk and Marcin Wrochna. On space efficiency of algorithms working on structural decompositions of graphs. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPIcs*, pages 57:1–57:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

35 Rahul Santhanam. On separators, segregators and time versus space. In *Computational Complexity, 16th Annual IEEE Conference on, 2001.*, pages 286–294, 2001.

36 Uwe Schöning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 410–414. IEEE Computer Society, 1999.

37 Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.

38 Leslie G. Valiant and Vijay V. Vazirani. NP is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47(3):85–93, 1986.

39 Magnus Wahlström. Abusing the tutte matrix: An algebraic instance compression for the K-set-cycle problem. In Natacha Portier and Thomas Wilke, editors, *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 - March 2, 2013, Kiel, Germany*, volume 20 of *LIPIcs*, pages 341–352. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

40 Ryan Williams. Inductive time-space lower bounds for sat and related problems. *computational complexity*, 15(4):433–470.

A Proof of Lemma 3.5

We will need to recall the 'Hash Down' lemma from [33], along with the relevant terminology.

Let C be a circuit on n variables $x_1, \ldots, x_n$ and let $S^C \subseteq \{0,1\}^n$ be the set of assignments to $x_1, \ldots, x_n$ satisfying C. Let $s \leq \lfloor \log |S^C| \rfloor - 2$, and assume $s > 0$. Let $\mathbf{desc}$ be an encoding of the description of C and let $m = |\mathbf{desc}(C)|$. Let

$$t = (t_{-n+1}, \ldots, t_{-1}, t_0, t_1, \ldots, t_{n-1}) \in \{0,1\}^{2n-1},$$

and let T_t be the Toeplitz matrix defined by t, i.e., $T_t(i,j) = t_{(j-i)}$ for $0 \leq i, j \leq n-1$. For $w \in \{0,1\}^s$ and $x \in \{0,1\}^{n-s}$, we define the column vector $(w;x)$ to be the concatenation of w and x. For $w \in \{0,1\}^s$ and t such that T_t is invertible, we define

$$J_{t,w}(x) := T_t^{-1} \cdot (w;x), \qquad \mathbf{Sparse}^{t,w}(C)(x) := C(J_{t,w}(x))$$

▶ **Lemma A.1** (Lemma 1 from [33]). *If t, w are picked uniformly and independently at random, C and $\mathbf{Sparse}^{t,w}(C)$ are equivalent with respect to satisfiability with probability at least $1/4$. Furthermore, $\mathbf{Sparse}^{t,w}(C)$ can be constructed in time $\tilde{O}(m)$ and its description can be computed by a circuit of size $\tilde{O}(m)$ given $\mathbf{desc}(C)$,t and w.*

Proof of Lemma 3.5. We start with the first equivalence. The forward implication follows easily since we can simply guess at random a satisfying assignment of the circuit C resulting from the witness compression.

For the backward direction, assume we have a randomized polynomial time algorithm V that solves Q with success probability at least $2^{-h(k)}$ (which is without loss of generality since we may dispense with factors inversely polynomial in the input size by using a polynomial number of independent trials). Let r be the number of random bits used by V. With standard techniques as used in Cook's theorem, we can transform this algorithm into a Boolean circuit C on r variables such that $C(y_1, \ldots, y_r)$ is equivalent to the outcome of running A on the given instance with random bits $y_1, \ldots, y_r$. It follows that the set $S^C \subseteq \{0,1\}^r$ of assignments to $y_1, \ldots, y_r$ satisfying C has the property $|S^C| \geq 2^r/2^{h(k)}$. Therefore, we may set $s = r - h(k) - 2$, pick t and w uniformly and independently at random and construct a circuit $\mathbf{Sparse}^{t,w}(C)$ using Lemma A.1 that is equivalent to C with respect to satisfiability with probability at least $1/4$. Now the claim follows as $\mathbf{Sparse}^{t,w}(C)$ is on at most $r - s = h(k) + 2$ variables, and an equivalent circuit on $h(k)$ variables can be found just by brute-forcing over assignments to the first 2 variables.

We continue with the second equivalence. Again, the forward implication follows directly since we can simply guess a satisfying assignment of the circuit C resulting from the witness compression. The proof of the backward direction is also very similar to the backward direction of the previous part, except now we need the fact that CKT-SAT is PC-complete and use Levin reductions to map the verifier V to a circuit that outputs a witness with probability $2^{-h(k)}$, before we apply Lemma A.1. ◀

B Some Constructive Polynomial Witness Compressions.

In this section we will demonstrate the power of polynomial witness compression by revisiting some known algorithms and observe that they give rise to (quasi)-linear witness compressions, thereby proving Theorem 3.6. We would like to stress that by no means these results are surprising or require any new methods[10], but anyway provide descriptions for

[10] And indeed, according to some anonymous reviewer these results are folklore.

completeness.

We study the following problems

LONG PATH **Parameter:** k.

Instance: Graph $G = (V, E)$ and integer k.

Witness: A simple path of length k.

STEINER TREE **Parameter:** k.

Instance: Graph $G = (V, E)$ and set of terminal vertices $K \subseteq V$ with $|K| = k$ and integer θ.

Witness: A set S with $K \subseteq S$ such that $G[S]$ is connected and $|S| \leq \theta$.

DIRECTED FEEDBACK VERTEX SET (DFVS) **Parameter:** k.

Instance: Directed graph $G = (V, E)$, integer k.

Witness: A set $X \subseteq E$ with $|X| \leq k$ such that $(V, E \setminus X)$ is acyclic.

We will provide algorithms that find solutions with sufficiently good probability; the statement of Theorem 3.6 follows directly from this by Lemma 3.5.

B.1 Longest Path

The *color-coding technique* directly applies here: Let $G = (V, E)$ and let $v_1, \ldots, v_k$ be the vertices visited in order by the simple path. For every vertex $v \in V$ assign a color $c(v) \in \{1, \ldots, k\}$ uniformly and independently at random. We see that

$$\Pr[\forall_{1 \leq i \leq k} c(v_i) = i] = \frac{1}{k^k},$$

and given such a coloring c, it can be determined in linear time whether a path with consecutive colors exists by a longest path computation in a directed acyclic graph.

B.2 Steiner Tree

Note that equivalently, we may look for a rooted tree $T = (S, E')$ with $K \subseteq S \subseteq V$ and $E' \subseteq E$ (to see this, simply pick E' to be a spanning tree of $G[S]$). Note also that we may assume that the set of leaves in this tree equals the set of terminals: We may replace every terminal k' in G with a vertex v and add k' as a pendant vertex to it so terminals must be in leaves, and if a leaf vertex in T is not a terminal we may always omit it.

We first guess the topology T' of the tree T obtained after vertices of degree two are contracted: By Cayley's formula [7], there are at most l^{l-2} different trees on l vertices and since the number of vertices of T' is at most $2k$, we have at most $2^{O(k \log(k))}$ possibilities. It is not hard to see that such trees can also be uniformly sampled in polynomial time.

Thus we may uniformly sample a tree T' on at most $2k$ vertices and restrict our attention to finding T whose topology is the given tree T'. To do this, use a polynomial time dynamic programming algorithm, based on the following definition and recurrence: for a vertex u of T' and a vertex $v \in V$ define $A[u, v]$ as

$$\min\{|E'| : T'[u] \text{ obtainable from } (S', E') \text{ by unifying } u, v \text{ and contracting degree-2 vertices}\}.$$

Then it is easy to see that

$$A[u, v] = \begin{cases} 0, & \text{if } u \text{ is a leaf in } T' \text{ and } u = v \\ \infty, & \text{if } u \text{ is a leaf in } T' \text{ and } u \neq v \\ \sum_{i=1}^{\ell} \min_{v' \in V}\{A[c_i, v'] + d(v, v')\}, & \text{if } u \text{ has children } c_1, \ldots, c_\ell \text{ in } T', \end{cases}$$

where $T[v']$ denotes the subtree of T' rooted at v and $d(v, v')$ denotes the number of edges on a shortest path from v to v' in G.

The answer can now be computed as $\min_{v \in V}\{A[u, v]\}$ where u is the root of T', and it is easy to in fact find a tree that implements this value.

B.3 Directed Feedback Vertex Set

In this section we revisit the proof of the result that DFVS/k is FPT. A large part of this subsection directly follows parts of Chapeter 8 from [11]. We follow the notation that for a graph G, $V(G)$ and $E(G)$ denote the vertex set and edge set respectively.

▶ **Definition B.1.** Let G be a directed graph. For disjoint sets $X, Y \subseteq V(G)$, an (X, Y)-cut is a set of edges $C \subseteq E(G)$ such that no vertex of Y is reachable from a vertex of X in $G \setminus C$.

▶ **Definition B.2.** Let G be a directed graph and let $X, Y \subseteq V(G)$ be disjoint sets of vertices. Let $S \subseteq E(G)$ be an (X, Y)-cut and let R be the set of vertices reachable from X in $G \setminus S$. We say that S is an *important* (X, Y)*-cut* if it is minimal and there is no (X, Y)-cut S' with $|S'| \leq |S|$ such that $R \subset R'$, where R' is the set of vertices reachable from X in $G \setminus S'$.

The following is a mild variant of [11, Theorem 8.36], and was observed[11] in [28, Lemma 1].

▶ **Theorem B.3.** *There is a randomized polynomial time algorithm that takes as input a directed graph G on n vertices and m edges, $X, Y \subseteq V(G)$ and an integer k, and outputs a set $S \subseteq E$ such that for every important (X, Y)-cut S' of size at most k, we have* $\Pr[S' = S] \geq 4^{-k}$.

Skew Edge Multicut **Parameter:** k.

Instance: Directed graph $D = (V, A)$, vertices $s_1, \ldots, s_\ell, t_1, \ldots, t_\ell \in V$ and integer k.

Witness: $S \subseteq A$ such that $|S| \leq k$ and $(V, E \setminus S)$ has no path from s_i to t_j, for any $i \geq j$.

To solve this problem, we require the following 'pushing lemma for Skew Edge Multicut', is Lemma 8.40 from [11]:

▶ **Lemma B.4.** *Let $(D, s_1, \ldots, s_\ell, t_1, \ldots, t_\ell, k)$ be an instance of* Skew Edge Multicut. *If the instance has a solution S, then it has a solution S^* with $|S^*| \leq |S|$ that contains an important $(s_\ell, \{t_1, \ldots, t_\ell\})$-cut.*

▶ **Theorem B.5.** *There is a polynomial time algorithm that finds solutions for* Skew Edge Multicut *with probability at least 4^{-k}.*

Proof. Suppose the given instance is a yes-instance. Let i be the largest integer such that there exists a path from s_i to t_j for some. If such an i does not exist, the instance is trivially solvable. By Lemma B.4, there exists a witness S^* that contains an important $(s_i, \{t_1, \ldots, t_\ell\})$-cut. Use Theorem **??** to guess such an important cut C and then output the union of this cut with the output of a recursive call on the graph $(V, E \setminus C)$ with parameter $k - |C|$. It is easy to see that this algorithm runs in (expected) polynomial time and outputs a solution with probability at least 4^{-k}. ◀

Directed Feedback Arc Set Compression **Parameter:** k.

[11] We thank an anonymous reviewer for pointing that this was explicitly proved in [28]

Instance: Directed graph $G = (V, E)$, Feedback Vertex Set $W \subseteq V$ and integer k.

Witness: A set $S \subseteq E$ such that $(V, E \setminus S)$ is acyclic.

▶ **Lemma B.6.** *There is a polynomial time algorithm that finds solutions for* Directed Feedback Arc Set Compression *with probability at least* $1/(|W|!4^k)$.

Proof. We uniformly guess one of the $|W|!$ orderings of $|W|$ and denote it with $w_1, \ldots, w_{|W|}$. Note that if a solution S exists, there must exist a topological ordering of $(V, E \setminus S)$ and the probability that such an ordering induces $w_1, \ldots, w_{|W|}$ on W is at least $|W|!$.

We build a graph in the following way: replace every vertex w_i with two vertices s_i, t_i and add an edge (t_i, s_i) between them. We define $E_W = \{(t_i, s_i) : 1 \leq i \leq |W|\}$. Then we replace every edge (a, w_i) with (a, t_i), every edge (w_i, a) with (s_i, a) and every edge (w_i, w_j) with (s_i, t_j). Let us define the Skew Edge Multicut instance as $(G', ((s_1, t_1), \ldots, (s_{|W|}, t_{|W|})), k)$. The following claims from Lemma 8.43 in [11] establish a connection between the solutions of this instance and our problem.

▶ **Claim B.7.** If there is a set $S \subseteq E(G)$ of size at most k such that $G \setminus S$ has a topological ordering inducing the order $w_1, \ldots, w_{|W|}$, then the Skew Edge Multicut instance has a solution.

▶ **Claim B.8.** Given a solution S' of the Skew Edge Multicut instance, one can find a feedback arc set S of size at most k for G in polynomial time.

The lemma follows by applying the Algorithm from Theorem B.5 on the obtained Skew Edge Multicut instance. ◀

Directed Feedback Arc Set (DFAS) **Parameter:** k

Instance: Directed graph $G = (V, E)$, integer k.

Witness: A set $S \subseteq A$ such that $(V, E \setminus S)$ is acyclic.

The following is due to Even et al. [18]:

▶ **Theorem B.9.** *There is a polynomial time algorithm that, given a directed graph with a feedback vertex set of size k, finds a feedback vertex set of size at most $c \cdot k \log(k) \log(\log(k))$, for some absolute constant c.*

Now we use the approximation scheme along the lines of Remark 5.3 in [8]:

▶ **Theorem B.10.** *There is a polynomial time algorithm that finds solutions for* Directed Feedback Arc Set Compression *with probability at least* $1/O(2^{k \log^3(k)} 4^k)$.

Proof. Use Theorem B.9 to find an approximate directed feedback vertex set W. If W is larger than $c \cdot k \log(k) \log(\log(k))$, we may return NO. Otherwise, we can directly use the algorithm of Lemma B.6. ◀

Now the main result from this section follows from an easy reduction:

Proof of Theorem 3.6. Given an instance G, k of DFVS, replace every vertex v with vertices v_{in} and v_{out} where v_{in} has as in-neighbors all in-neighbors of v and v_{in} has as out-neighbors all out-neighbors of v and there is an arc from v_{in} to v_{out} to obtain graph G'. It is easy to see that (G', k) is a yes-instance of DFAS if and only if (G, k) is a yes-instance of DFVS. ◀

C Proof of Theorem 3.2

The proof of this theorem crucially relies on Theorem 2.1. We will use the following straightforward lemma that was already used for a similar goal in [15]:

▶ **Lemma C.1** ([15]). *There is a deterministic polynomial algorithm that takes as input descriptions of 3CNF formulas $\varphi_1, \ldots, \varphi_b$ on disjoint sets of variables and an integer $c \geq 0$ and returns a 3CNF formula φ' that is satisfiable if and only if at least c of the φ_i's are satisfiable. Moreover, φ' contains designated variables $s_1, \ldots, s_b$ such that **1.** every satisfying assignment of φ' sets at least c of the variable $s_1, \ldots, s_b$ to 1, **2.** and for each $i \in [b]$, if φ' has a satisfying assignment with $s_i = 1$ then φ_i is satisfiable. If φ_i has m_i clauses then φ' has $O(b + \sum_{i \in [b]} m_i)$ clauses.*

Before we proceed with the proof, let us restate the theorem here for convenience:

▶ **Theorem 3.2** (restated). If there is a constructive AND-composition of degree d from a PC-hard search problem L into a parameterized search problem Q, then no polynomial time algorithm finds solutions for every instance (x, k) of Q with probability $\exp(-\text{poly}(k)|x|^{1/d-\Omega(1)})$, unless $NP \subseteq coNP/poly$.

Proof. Let $f = \chi_{\text{3CNF-SAT}}$ with respect to some fixed encoding of 3CNF-formulas. Since 3CNF-SAT is NP-complete and we assume $NP \not\subseteq coNP/poly$ we see that $f \notin NP/poly \cap coNP/poly$. Thus, by Theorem 2.1, it is sufficient to show that if a polynomial time algorithm with success probability as stated exists, for some polynomially bounded function $t(N)$ we can construct an algorithm A such that for any N:

1. A accepts $x_1, \ldots, x_{t(N)} \in \{0,1\}^N$ as input,
2. A outputs in $\text{poly}(\sum_{j=1}^{t} x_j)$ time t bits $z_1, \ldots, z_t \in \{0,1\}$,
3. $\Pr[\forall_{i=1}^{t} : x_i \in \text{3CNF-SAT} \leftrightarrow z_i = 1] > \exp(-o(t(N)))$.

The remainder of this proof is devoted to construct such an algorithm. Let $t = t(N)$, and let φ_i be the 3CNF-formula represented by x_i. The algorithm proceeds as follows:

1. Arbitrarily partition $x_1, \ldots, x_t$ into $\ell = \lceil t/b \rceil$ blocks $B_1, \ldots, B_\ell$ of size at most b each.
2. For every $i = 1, \ldots, \ell$ do the following:
   a. Uniformly pick an integer $c_i \in [b]$.
   b. Apply Lemma C.1 with formulas $\{\varphi_j\}_{j \in B_i}$ and $c = c_i$ to obtain a formula φ'_i.
   c. Use the Levin reduction implied by the PC-completeness[12] of L to transform φ'_i to an instance x'_i of L.
3. Run the assumed constructive AND-composition of degree d from L into Q on the set of instances $x'_1, \ldots, x'_\ell$ to obtain an instance (x, k) of Q.
4. Run the assumed algorithm to find with probability $\exp(-\text{poly}(k)|x|^{1/d-\Omega(1)})$ a string y such that $(x, k, y) \in Q$, if it exists.
5. Run algorithm B of the constructive AND-composition to retrieve y'_i such that

$$\Pr[\forall i : (x'_i, y'_i) \in L] \geq \exp\left(-\text{poly}\left(\max_i |x'_i|\right) \log(t)\right).$$

6. For every $i = 1, \ldots, \ell$ do the following:

[12] Confer Definition 2.8 of [20].

a. Verify whether $(x'_i, y'_i) \in L$ and if not, abort.
b. Use the second part of the Levin reduction from Step 2.c to obtain from y'_i a satisfying assignment of φ'_i.
c. Let $s_{i,j}$ be the value of the designated variable as promised in Lemma C.1 for x_j in this assignment.
d. For every $x_j \in B_i$, set $z_j = s_{i,j}$.

7. Return $z_1, \ldots, z_t$.

For verifying whether this implements algorithm A as needed, note it trivially satisfies Property 1. and 2.. For Property 3., denote c_i^* for the number of satisfiable instances in B_i. Suppose that $c_i = c_i^*$ for every i and both the algorithm at Line 4. and the constructive AND-composition at Line 5. are successful. Then y'_i is a satisfying assignment of φ'_i for every i and by Lemma C.1 the output $z_1, \ldots, z_t$ indeed satisfies $x_i \in$ 3CNF-SAT $\leftrightarrow z_i = 1$ for every $1 \leq i \leq 1$. Thus, we see that $\Pr[\forall_{i=1}^t : x_i \in \text{3CNF-SAT} \leftrightarrow z_i = 1]$ is at least

$$\Pr[\forall_i : c_i = c_i^*] \Pr[(x,k,y) \in Q | \forall_i : c_i = c_i^*] \Pr[\forall i : (x'_i, y'_i) \in L | (x,k,y) \in Q, \forall i : c_i = c_i^*],$$

and by the guarantee on the success probability of the algorithm of Line 4. and the constructive AND-composition we can lower bound this by

$$\geq \Pr[\forall_i : c_i = c_i^*] \exp(-\text{poly}(k)|x|^{1/d-\Omega(1)}) \exp\left(-\text{poly}\left(\max_i |x'_i|\right) \log(t)\right).$$

By the definition of constructive AND-composition we see that k is $\text{poly}(\max_i |x_i| \log(t))$ and $|x|$ is at most $\text{poly}(\max_i |x_i|) t^d$, and computing the first probability using that the c_i are picked uniformly and independently at random we can continue the derivation with

$$\begin{aligned}
&\geq b^{-t/b} \exp\left(-\text{poly}\left(\max_i |x_i| \log(t)\right) \left(\text{poly}(\max_i |x_i| \log(t)) t^d\right)^{1/d-\Omega(1)} - \text{poly}\left(\max_i |x'_i|\right) \log(t)\right) \\
&\geq b^{-t/b} \exp\left(-\text{poly}\left(\max_i |x_i| \log(t)\right) (t^d)^{1/d-\Omega(1)} - \text{poly}\left(\max_i |x_i| b\right) \log(t)\right) \\
&\geq b^{-t/b} \exp\left(-\text{poly}\left(\max_i |x_i| \log(t)\right) t^{1-\Omega(d)} - \text{poly}\left(\max_i |x_i| b\right) \log(t)\right).
\end{aligned}$$

Recall that d is a constant given by the constructive AND composition. Now it is clear that by first choosing b to be arbitrarily large, and subsequently $t = t(N)$ to be a polynomial large enough such that the $t^{\Omega(d)} = t^{\Omega(1)}$ term dominates the terms $\text{poly}(\max_i |x_i| \log(t))$ and $\text{poly}(\max_i |x_i| b) \log(t)$, we are able to obtain probability at least $\exp(-t/C)$ for any C, which is equivalent to $\exp(o(-t))$ achieving the desired contradiction. ◀

D Parity Compression

In this section we show that some algorithms can be seen as parity compressions, similarly as some algorithm can be seen as witness compressions. Recall that parity compressions are reductions to the following problem:

$\oplus$CKT-SAT **Parameter:** n

Instance: A Boolean circuit C on n variables

Asked: The parity of the size of the set $\{x \in \{0,1\}^n : C(x) = 1\}$ is odd.

Since we would like to show that many recent interesting algorithms using arithmetic in $GF(2^\ell)$ also are parity compressions, we will also study the following version of $\oplus$CKT-Sat:

$\oplus^\ell$CKT-SAT **Parameter:** n

Instance: A Boolean circuit on n variables computing a function $C : \{0,1\}^n \to GF(2^\ell)$, with ℓ polynomial in n.

Asked: Whether $\sum_{x\in\{0,1\}^n} C(x) \neq 0$.

The following follows quickly from the standard embedding of $GF(2^\ell)$:

▶ **Theorem D.1.** *There is a polynomial time algorithm that takes an instance $C : \{0,1\}^n \to GF(2^\ell)$ of $\oplus^\ell$CKT-Sat as input and outputs an instance $C : \{0,1\}^n \to \{0,1\}$ of $\oplus$CKT-Sat that is equivalent with one-sided constant error probability (e.g., with only false positives), with $|C'| \leq |C|\text{poly}(\ell)$.*

Proof. It is well known that an element a of $GF(2^\ell)$ can be represented as a degree ℓ polynomial $a_1 + a_2x + \ldots + a_\ell x^{\ell-1} \in \mathbb{Z}_2[x]$. Using circuit C, we can construct a circuit C_i such that $C_i(x) = a_i$ if $C(x) = a_1 + a_2x + \ldots + a_\ell x^{\ell-1}$.

Based on this, we construct C' as follows: pick a random subset $X \in \{1,\ldots,l\}$ and construct C' such that $C'(x)$ is the parity of $\sum_{i=1}^{\ell} C_i(x)$. By the random subsum principle (e.g., Section 11.5.1 in [3]), we see that if $\sum_{x\in\{0,1\}^n} C(x) = 0$, $\sum_{x\in\{0,1\}^n} C'(x) = 0$ and otherwise $\sum_{x\in\{0,1\}^n} C'(x) = 1$ with probability $1/2$. ◀

D.1 Long Path

In this section we briefly revisit the Long Path, and briefly outline how the existing fastest algorithm is captured as parity compression. Recall the problem statement of Long Path:

Long Path **Parameter:** k.

Instance: Graph $G = (V, E)$ and integer k

Witness: A simple path of length k (a so-called k-path)

In page [11, Page 350], a polynomial R depending on G (and random bits) is defined that is not identical to the zero polynomial with probability $\Omega(k^{-1})$ if G contains a k-path (Corollary 10.27), and identical to zero if it does not contain a k-path. As argued on page [11, Page 351], this polynomial can be evaluated in $O^*(2^{3k/4})$ time by inclusion-exclusion over the set L which satisfies $|L| \leq 3k/4$ since this gives a summation of $O^*(2^{|L|})$, each of which summands can be computed in polynomial time. Based on this we can reduce this computation to $\oplus^\ell$CKT-Sat on $\log(|L|)$ variables, and thus also to $\oplus$CKT-SAT by Theorem D.1.

D.2 K-Cycle

In this section we revisit an approach by Wahlström for the K-cycle problem [39] and note his approach can be seen as a parity compression as well. The problem is defined as follows:

K-Cycle **Parameter:** k.

Instance: Graph $G = (V, E)$, $K \subseteq V$, $k = |K|$

Witness: A cycle visiting all vertices from K.

For clarity, we'll restate the following two results:

▶ **Lemma D.2** ([39], Lemma 2). *Let $P(x_1,\ldots,x_n)$ be a polynomial of a field of characteristic two, and $T \subseteq [n]$ a set of target indices. For a set $I \subseteq [n]$, define $P_{-I}(x_1,\ldots,x_n) = P_{-I}(y_1,\ldots,y_n)$ where $y_i = 0$ for $i \in I$ and $y_i = x_i$ otherwise. Define*

$$Q(x_1,\ldots,x_n) = \sum_{I\subseteq T} P_{-I}(x_1,\ldots,x_n).$$

Then for any monomial m such that $t := \prod_{i \in T} x_i$ divides m we have $Q(m) = P(m)$, and for every other monomial we have $Q(m) = 0$.

▶ **Theorem D.3** ([39], Theorem 1). *Let $T = \{x_{i,k+2i-1}, x_{i,k+2i-1} : i \in [k]\}$ and $t = \prod_{x \in T} x$. Then G has a K-cycle if and only if $det(M_G)$, viewed as a polynomial, contains a monomial m with non-zero coefficient such that t divides m.*

Here we refer to [39] for the definition of M_G and merely note it can be computed in polynomial time. Then Wahlström continues by observing that Theorem D.3 implies an $O^*(4^{|K|})$ time algorithm for K-CYCLE since after applying Lemma D.2 we may evaluate the resulting polynomial Q randomly over $GF(2^\ell)$ for $\ell = \Omega(\log n)$. Here we stress that the polynomial $(\det(M_G))_{-I}$ can be computed in polynomial time. Thus the algorithm can be seen as a reduction to the $\oplus^\ell$CKT-SAT problem where the associated circuit has an indicator variable for each element of T and computes $(\det(M_G))_{-I}$ for the subset I as represented by the input bits of the circuit.

E Disjunctive Dynamic Programming.

Recall that the problem central to disjunctive dynamic programming is:

CNF-REACH **Parameter:** n

Instance: A CNF-formula $\varphi : \{0,1\}^n \to \{0,1\}$ with m clauses and even n even, integer $\ell = \text{poly}(n)$.

Witness: $x^1, \ldots, x^\ell \in \{0,1\}^{n/2}$ with $x^1 = 00\cdots0$,$x^\ell = 11\cdots1$ and $\varphi(x^i x^{i+1}) = 1$ for every $0 \leq i \leq \ell - 1$.

For ease of showing an algorithm is a disjunctive dynamic program, we show that CNF-REACH is equivalent to the following generalization:

CNF-REACH' **Parameter:** n

Instance: CNF-formula's $\varphi^1, \ldots, \varphi^{\ell-1} : \{0,1\}^n \to \{0,1\}$ with m clauses and even n each, integer ℓ.

Witness: $x^1, \ldots, x^\ell \in \{0,1\}^{n/2}$ with $x^1 = 00\cdots0$,$x^\ell = 11\cdots1$ and $\varphi_i(x^i x^{i+1}) = 1$ for every $0 \leq i \leq \ell - 1$.

To see how to reduce from CNF-REACH' to CNF-REACH, note we can add $\lceil \log(\ell) \rceil$ variables representing ℓ and use

$$\varphi = \bigwedge_{i=1}^{\ell-1} (\varphi_i \vee [\ell \neq i]) .$$

It is easily seen that, when rewritten as a CNF formula using distributivity, φ has at most $m\ell$ clauses (and there sizes have increased with $\lceil \log(\ell) \rceil$ each).

E.1 IS/pw is equivalent to CNF-Reach

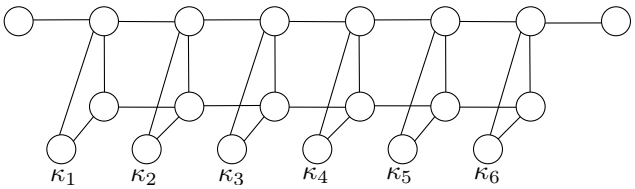
In this subsection we provide to (nearly) tight reductions between IS/pw and CNF-Reach.

IS/pw is at least as hard as CNF-Reach

▶ **Theorem E.1.** *There exist a function f and polynomial time algorithm that given $\epsilon > 0$, CNF-formula φ on n variables and m clauses and integer ℓ, outputs an integer θ and graph G on at most $f(\epsilon)nm\ell$ vertices along with a path decomposition of G of width $(1+\epsilon)n + f(\epsilon)$*

such that (φ, ℓ) *is a yes-instance of* CNF-Reach *if and only if* G *has an independent set of size at least* θ'.

Figure 1 The graph M_6



▶ **Definition E.2.** The graph M_r is the graph consisting of a sequence of r triangles with each two consecutive ones connected by two edges, and two pendant vertices attached to the first and the last triangle. We denote $\kappa_1, \ldots, \kappa_r$ for the degree-2 vertices in the triangles numbered consecutively.

An example for $r = 6$ is provided in Figure 1.

▶ **Claim E.3** (Claim 14.39 [11])**.** Assume r is even. Then the maximum size of an independent set in M_r is equal to $r+2$, and each independent set of this size contains vertex κ_i for some i. Moreover, for every $z = 1, 2, \ldots, r$ there exists an independent set Z_z in M_r such that $|Z_z| = r+2$ and $Z_z \cap \{\kappa_1, \ldots, \kappa_r\} = \{\kappa_z\}$. Finally, M_r has pathwidth at most 3.

Proof of Theorem E.1. Let φ be a CNF-formula in n variables and m clauses. We will now construct a graph $G(\varphi)$ that encodes φ in some sense; this will function as a gadget used for the construction of G. Let a be an even integer dividing $n/2$ to be set later (after a is set, we can assure a divides $n/2$ by adding few dummy variables to φ). Let $b = a + 2\lceil \log a \rceil$. Note that

$$\binom{b}{b/2} \geq 2^b/b \geq \frac{2^a a^2}{a + 2\lceil \log a \rceil} \geq 2^a. \tag{E.1}$$

Let $g = \lceil \frac{n}{a} \rceil$ and partition $[n]$ in g blocks A_h of size a for $h = 1, \ldots, g$. Similarly, denote $p = bg$ and partition $[p]$ into g blocks B_h with $|B_h| = b$.

For $h = 1, \ldots, g$, arbitrarily fix an injective function $\eta_h : 2^{A_h} \to \binom{B_h}{b/2}$ (which is possible by virtue of (E.1)). Note this naturally induces an injective encoding $\eta : \{0,1\}^n \to \{0,1\}^p$ of assignments of φ to binary strings of length p.[13] Now construct the graph $G(\varphi)$ as follows:

- For every $i = 1, \ldots, p$ and $j = 1, \ldots, 2m$ add a vertex p_{ij}; for $1 \leq i \leq p$ and $j = 1, \ldots, 2m-1$ add an edge $(p_{ij}, p_{i+1,j})$.
- For every $j = 1, \ldots, 2m$ and $h = 1, \ldots, g$ do the following:
  - For every $X \in \eta_h(2^{A_h})$, add a vertex z_X^{hj} and make it adjacent to all p_{ij} with $j \in B_h \backslash X$.
  - Add edges between z_X^{hj} and z_Y^{hj} for every $X, Y \in \eta_h(2^{A_h})$.
- For every $j = 1, 3, 5, \ldots, 2m-1$ do the following
  - Denote $S_j = \left\{ X \in \binom{B_h}{b/2} : 1 \leq h \leq \lceil g \rceil \wedge \eta^{-1}(X) \text{ satisfies } C_{(j+1)/2} \right\}$.

[13] In this proof we freely interpret subsets of U as binary vectors indexed by U (e.g., elements of $\{0,1\}^U$ in the natural way.

- If $|S_j|$ is odd, pick an arbitrary element of S_j and add it again to S_j to assure $|S_j|$ is even (as a multiset).
- Add a copy M_r^j of the graph M_r, where $r = |S_j|$. Label the degree-2 vertices in the triangles of this copy with κ_X^j for $X \in S_j$ in order of increasing h.
- For every $X \in S_j$ do
  - Let h be such that $X \subseteq B_h$.
  - Add edges between p_{ij} and κ_X^j such that $i \in B_h \setminus X$.

For a set I of vertices of $G(\varphi)$, denote $y(j, I) \in \{0,1\}^p$ the vector with the $i$þcoordinate being 1 if $p_{ij} \in I$ and 0 otherwise.

▶ **Claim E.4.** φ has a satisfying assignment $x \in \{0,1\}^n$ if and only if $G(\varphi)$ has an independent set of size $\theta = m(p + 2g + 2) + \sum_{j=1}^{m} |S_j|$. Moreover, for any independent set I of size θ, $\eta^{-1}(y(j_0, I)) = \eta^{-1}(y(j_1, \overline{I}))$ satisfies φ for every even j_0 and odd j_1. Here $\overline{I}$ denotes all vertices of G not in I.

Proof. Let us start with obtaining an independent set from a satisfying assignment: let $x \in \{0,1\}$. Construct I as follows:

1. For every j, do the following:
   - **a.** Include p_{ij} in I if $i \in \eta(x)$ and j is odd and include p_{ij} in I if $i \notin \eta(x)$ and j is even.
   - **b.** For every $h = 1, \ldots, g$, do the following:
     - **i.** Include z_X^{hj} in I where $X = \eta_h(x^h)$ and x^h denotes x restricted to B_h if j is odd.
     - **ii.** Include z_X^{hj} in I where $X = B_h \setminus \eta_h(x^h)$ and x^h denotes x restricted to B_h if j is even.
   - **c.** If j is odd, note that since x satisfies clause C_{2j-1} there exists some $x_i \in A_h$ satisfying C_{2j-1}. Therefore, if x^h denotes x restricted to B_h, $\eta_h(x_h) \in S_j$ and hence κ_X^j is not adjacent to any of the vertices added to I. The assumption of the second part of Claim E.3 applies to M_r^j.
   - **d.** Add an independent set of M_r^j of size $|S_j| + 2$ as given by Claim E.3.

Since in $1(a)$ we add pm vertices, in $1(b)$ we add $2mg$ vertices and in $(1)d$ we add $\sum_{j=1}^{m}(|S_j|+2)$ vertices to I, and I is an independent set by the definition of $G(\varphi)$, this direction follows.

Now we show how to obtain a satisfying assignment from an independent set I of size θ: first note that the vertex set of $G(\varphi)$ can be covered with p paths of length $2m$, $2gm$ cliques and copies of M_r^j. By the simple upper bounds on the independent sets in these graphs, we see that I needs to take at least half of the vertices from the p paths formed by the vertices p_{ij}, at least one vertex of every clique formed by the vertices z_X^{hj} and $r + 2$ vertices from every copy of M_r^j.

Moreover, by the construction of the vertices z_X^{hj} we see that I needs to take exactly half of the vertices $\{p_{ij} : i \in B_h\}$ for every h, j: if it takes more no vertex z_X^{hj} can be included and if it takes less it needs to take more than half of the vertices elsewhere in paths which is also not possible by the same argument. Thus, I needs to include exactly one of the vertices $p_{ij}, p_{i,j+1}$ for every $i = 1, \ldots, p$ and $j = 1, \ldots, 2m - 1$. This implies that $\eta^{-1}(y(j_0, I)) = \eta^{-1}(y(j_1, \overline{I}))$ for every even j_0 and odd j_1.

In order to see that this uniquely encoded assignment of x in fact satisfies φ, consider the column j where a clause is $C_{(j+1)/2}$ is handled: since I needs to include at least $|S_j| + 2$ vertices from M_r^j, by Claim E.3 I includes κ_X^{hj} for some X. This means by construction that x restricted to B_h satisfies clause C_{2j-1}. ◀

▶ **Claim E.5.** $G(\varphi)$ has a path decomposition with all vertices $p_{i,1}$ in the first bag and $p_{i,m}$ in the last bag of width $p + 2^a$ that can be found in polynomial time, for some function f.

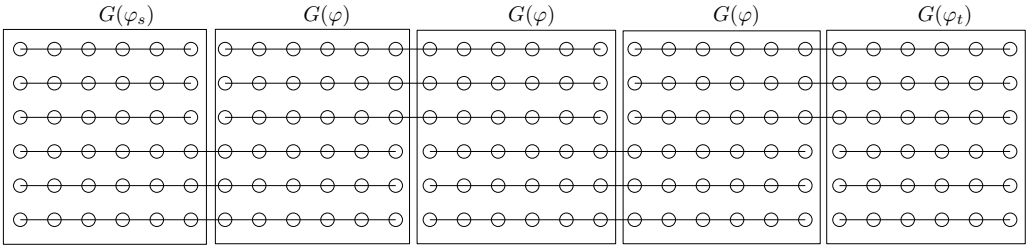
Proof. The path decomposition has a bag B_i containing the set $\{p_{ij} : 1 \leq i \leq p\}$ for every j. Note that these sets are indeed separators in $G(\varphi)$, and for odd j we may simply cover all vertices z_X^{hj} and the graph M_r^j since the graph induced by these vertices has pathwidth at most 2^p. To cover the edges of the type $(p_{ij}, p_{i+1,j})$ we make $2p$ bags interatively introducing $p_{i,j+1}$ and forgetting $p_{i,j}$. ◀

Now the graph implementing the theorem statement is constructed as follows:

1. Let $\varphi_s = \varphi \wedge x_1 = 0 \wedge \ldots \wedge x_{n/2} = 0$ and $\varphi_s = \varphi \wedge x_{n/2+1} = 1 \wedge \ldots \wedge x_n = 1$. Construct graphs $G_1, \ldots, G_\ell$ with $G_1 = G(\varphi_s)$, $G_\ell = G(\varphi_t)$ and $G_k = G(\varphi)$ for $1 < k < \ell$.
2. For every $k = 2, \ldots, \ell$ do the following:
   - **a.** Denote l_i for the vertex $p_{i,2m}$ in G_{k-1}, and f_i for the vertex $p_{i,1}$ in G_k[14].
   - **b.** If k is odd, add edges between l_i and f_i for every $p/2 < i \leq p$,
   - **c.** If k is even, add edges between l_i and f_i for every $1 \leq i \leq p/2$.

The threshold θ' is set to $\ell\theta$. An illustration is provided in Figure 2.

Figure 2 Overview of the construction.



By Claim E.4 we see that if G has an independent set of size θ' (the complement of) this independent set needs to encode a satisfying assignment of the relevant CNF-formula in the (last) first columnn of G_k, via the encoding η. Moreover, since I includes exactly half of each set $\{p_{ij} : 1 \leq i \leq p\}$ for every j, this means that for even k the intersection of the independent set with the first $p/2$ elements of the odd columns in G_{k-1} and G_k encodes the same assignment in $\{0,1\}^{n/2}$, and similar correspondence occurs for odd k with the last $p/2$ elements of a column. Thus, the independent set of G indeed implies the existence of the sequence $x^1, \ldots, x^\ell$ as asked in the CNF-Reach problem.

For the other direction, note that we can use x^k, x^{k+1} to obtain a sufficiently large independent set in G_k from Claim E.4 and these independent sets will not share incident edges by the property of how they intersect with odd and even columns of G_k.

For the path decomposition of G, we may use the path decompositions of G_k obtained via Claim E.5. This can be concatenated in a natural way to obtain a path decomposition of G of width at most $p + 2^a$. Setting a large enough such that $\frac{2\lceil \log a \rceil}{a} < \epsilon$, we see that $p + 2^a = bg + 2^a \leq (a + 2\lceil \log(a) \rceil)(n/a + 1) + 2^a \leq (1 + \epsilon)n + f(\epsilon)$. ◀

[14] We'll later refer to such vertices as *first* and *last* vertices of G_{k-1} and G_k in the natural way.

E.1.1 IS/pw Reduces to CNF-Reach

E.1.1.1 Pathwidth and path decompositions.

A *path decomposition* of a graph $G = (V, E)$ is a path P in which each node x has an associated set of vertices $B_x \subseteq V$ (called a *bag*) such that $\bigcup B_x = V$ and the following properties hold:

1. For each edge $\{u, v\} \in E(G)$ there is a node x in P such that $u, v \in B_x$.
2. If $v \in B_x \cap B_y$ then $v \in B_z$ for all nodes z on the (unique) path from x to y in P.

The *pathwidth* of P is the size of the largest bag minus one, and the pathwidth of a graph G is the minimum pathwidth over all possible path decompositions of G. Since our focus here is on dynamic programming over a path decomposition we only mention in passing that the related notion of treewidth can be defined in the same way, except for letting the nodes of the decomposition form a tree instead of a path.

It is common for the presentation of dynamic programming to use path and tree decompositions that adhere to some simplifying properties, in order to make the description easier to follow. The most commonly used notion is that of a nice tree decomposition, introduced by Kloks [24]; the main idea is that adjacent nodes can be assumed to have bags differing by at most one vertex (this can be achieved without increasing the treewidth). For an overview of tree decompositions and dynamic programming on tree decompositions see [5, 23]. In a similar way, but using also the *introduce-edge bags* from [12], we define nice path decompositions as follows.

▶ **Definition E.6** (Nice Path Decomposition)**.** A *nice path decomposition* is a path decomposition where the underlying path of nodes is ordered from left to right (the predecessor of any node is its left neighbor) and in which each bag is of one of the following types:

- **First (leftmost) bag**: the bag associated with the leftmost node x is empty, $B_x = \emptyset$.
- **Introduce-vertex bag**: an internal node x of P with predecessor y such that $B_x = B_y \cup \{v\}$ for some $v \notin B_y$. This bag is said to *introduce* v.
- **Introduce-edge bag**: an internal node x of P labeled with an edge $\{u, v\} \in E(G)$ with one predecessor y for which $u, v \in B_x = B_y$. This bag is said to *introduce* uv.
- **Forget bag**: an internal node x of P with one predecessor y for which $B_x = B_y \setminus \{v\}$ for some $v \in B_y$. This bag is said to *forget* v.
- **Last (rightmost) bag**: the bag associated with the rightmost node x is empty, $B_x = \emptyset$.

It is easy to verify that any given path decomposition of pathwidth $\mathtt{pw}$ can be transformed in time $|V(G)|\mathtt{pw}^{O(1)}$ into a nice path decomposition without increasing the width.

E.1.1.2 A folklore dynamic programming algorithm.

For a node x, denote G_x for the subgraph of G induced by all vertices and edges introduced to the left of x in the path decomposition. For a node x, integer j and subset $Y \subseteq B_x$ define

$$T(x, i, Y) = \begin{cases} true, & \text{if there exists an independent set } I \text{ of } G_x \text{ satisfying } |I| \geq i \text{ and } I \cap B_x = Y \\ false, & \text{otherwise.} \end{cases}$$

We see that

$$T(x,i,Y) = \begin{cases} true & \text{if } x = 0,\ i \geq 0, \text{ and } Y = \emptyset, \\ false & \text{if } x = 0 \text{ and } Y \neq \emptyset \text{ or } i < 0, \\ T(x-1, i-1, Y \setminus v), & \text{if } x \text{ introduces } v \text{ and } v \in Y, \\ T(x-1, i, Y), & \text{if } x \text{ introduces } v \text{ and } v \notin Y, \\ T(x-1,i,Y) \vee T(x-1,i,Y \cup v), & \text{if } x \text{ forgets } v, \\ T(x-1,i,Y) & \text{if } x \text{ introduces } uv \text{ and } \{u,v\} \not\subseteq Y, \\ false, & \text{if } x \text{ introduces } uv \text{ and } u,v \in Y. \end{cases} \tag{E.2}$$

And G has an independent set of size at least θ if and only if $T(x,\theta,\emptyset) = true$, where x is the rightmost bag.

E.1.1.3 Translating to CNF-Reach'.

To translate the dynamic programming in the previous subsection to a reduction to CNF-Reach', we simply need to show that the adjacency matrix of each layer of the configuration graph of this dynamic programming recurrence can be encoded as small enough CNF formula. To see that this is possible, note we have a layer for every x, and for each layer there is a vertex for every triple (x,i,Y). There is an edge from $(x-1,i',Y')$ to (x,i,Y) if $T(x-1,i',Y')$ implies $T(x,i,Y)$ according to (E.2).

Thus for $x = 1,\ldots,\ell-1$ we may use $\varphi_x((i',Y'),(i,Y)) =$

$$\begin{cases} ([i' = i-1] \wedge [Y' = Y \setminus v] \wedge [v \in Y]) \vee ([i = i'] \wedge [Y' = Y \setminus] \wedge [v \notin Y]), & \text{if } x \text{ introduces } v, \\ ([i' = i] \wedge [Y = Y']) \vee ([i' = i] \wedge [Y = Y' \cup v]), & \text{if } x \text{ forgets } v, \\ ([i' = i] \wedge [\{u,v\} \not\subseteq Y]) & \text{if } x \text{ introduces } uv. \end{cases}$$

Note that if i', i are encoded in binary and Y, Y' are encoded with at most $\mathtt{pw}$ bits each indicating whether possible elements are included, in each of the three cases φ_x is easily written as a CNF formula on $2\mathtt{pw} + O(\log(n))$ variables and $\mathrm{poly}(\mathtt{pw}, \log(n))$ clauses. Moreover, it is easily seen for the first bag, the bit string 0000 indeed encodes the only state that is *true* in (E.2), and we can define φ_ℓ such that it is only satisfied by $((i,\emptyset), 11111)$ for $i \leq \theta$.

Combining this with the observation that CNF-Reach' reduces to CNF-Reach, we have shown:

▶ **Theorem E.7.** *There is a polynomial time reduction that, given an instance of* Independent Set*, outputs an equivalent instance of* CNF-Reach *with* $\ell = O(n\mathtt{pw})$*, and* φ *has at most* $2\mathtt{pw} + O(\log(n))$ *variables and* $\mathrm{poly}(\mathtt{pw}, \log(n))$ *clauses.*

E.1.2 Set-Cover Reduces to CNF-Reach.

In the Set Cover problem, we are given sets $S_1,\ldots,S_m \subseteq U$, integer θ seek to find a subset X satisfying $\cup_{i \in X} S_i = U$ and $|X| \leq \theta$. A standard dynamic programming algorithm is

known that uses the following recurrence:

$$T(x,i,Y) = \begin{cases} true & \text{if } x = 0,\ i \geq 0 \text{ and } Y = \emptyset, \\ false & \text{if } x = 0 \text{ and } Y \neq \emptyset \text{ or } i < 0, \\ T(x-1,i,Y) \vee T(x-1,i-1,Y \setminus S_x), & \text{otherwise.} \end{cases} \quad \text{(E.3)}$$

And we have a set cover if and only if $T(m,\theta,U) = true$.

As before, we define a layered graph with a layer for $x = 1,\ldots,m$. In every layer we have a vertex for each triple (x,i,Y) and edges $((x-1,i',Y'),(x,i,Y))$ if $T(x-1,i',Y')$ implies $T(x,i,Y)$. We see that we can use

$$\varphi_x((i',Y'),(i,Y)) = ([i = i'] \wedge [Y = Y']) \vee ([i' = i-1] \wedge [Y' = Y \setminus S_x]),$$

and similarly as in the previous subsection we conclude that

▶ **Theorem E.8.** *There is a polynomial time reduction that, given an instance of* Set Cover*, outputs an equivalent instance of* CNF-Reach *with* $\ell = m$*, and* φ *has at most* $2n + O(\log(m))$ *variables and* $\mathrm{poly}(n, \log(m))$ *clauses.*

F Missing Problem Definitions (mainly for Section 6)

Subset Sum **Parameter:** $\log(W)$.
Instance: Integers $w_1,\ldots,w_n$ and W
Witness: A subset $X \subseteq [n]$ such that $\sum_{e\in X} w_e = W$.

Knapsack **Parameter:** $\log(V)$.
Instance: Integers $v_1,\ldots,v_n,w_1,\ldots,w_n$ and V,W
Witness: A subset $X \subseteq [n]$ such that $\sum_{e\in X} w_e \leq W$ and $\sum_{e\in X} v_e \geq V$.

Knapsack/Weight-Value **Parameter:** $\log(V) + \log(W)$.
Instance: Integers $v_1,\ldots,v_n,w_1,\ldots,w_n$ and V,W
Witness: A subset $X \subseteq [n]$ such that $\sum_{e\in X} w_e \leq W$ and $\sum_{e\in X} v_e \geq V$.

K-Cycle **Parameter:** k.
Instance: Graph $G = (V,E)$, $K \subseteq V$, $k = |K|$
Witness: A cycle visiting all vertices from K.

Disjoint Paths **Parameter:** k.
Instance: Graph $G = (V,E)$, vertices $s_1,\ldots,s_k,t_1,\ldots,t_k$
Witness: A set of disjoint paths connecting s_i to t_i for every i.

G Proof of Theorem 3.3

We use the following Theorem:

▶ **Theorem G.1** ([26]). *Let G be an n-vertex graph with non-negative vertex costs summing to no more than one and let $0 \leq \epsilon \leq 1$. Then there is some set C of $O(\sqrt{n/\epsilon})$ vertices whose removal leaves G with no connected components of cost exceeding ϵ. Furthermore the set C can be found in $O(n \log n)$ time.*

Apply Theorem G.1. Assign to every vertex cost $1/n$ and set with $\epsilon = \log(n)/n$. We obtain a set C of size $O(\sqrt{n^2/\log(n)})$ which is $O(n/\sqrt{\log(n)})$ such that after removing C all connected components consist of at most $\log(n)$ vertices. Therefore we can simply guess $Y = C \cap X$ of a maximum independent set X by letting Y be a random subset from C and Y will be a correct guess with probability at least $1/2^{|C|}$ which is at least $1/2^{O(n/\sqrt{\log(n)})}$. Given a guess Y we can compute the maximum size of an independent set M such that $M \cap C = Y$ since after removing Y and its neighbors and C all connected components are of size $\log(n)$ and we can find the maximum independent set in each component independently by trying all at most $2^{\log(n)} = \text{poly}(n)$ subsets.

H Proof of Theorem 3.9

We give a simple constructive AND-composition from Independent Set to IS/pw: Given instances $(G_1, \theta_1), \ldots, (G_t, \theta_t)$ let G be the graph with a connected component G_i for every i and pick θ uniformly at random from $1, \ldots, |V(G)|$. With probability $1/|V(G)| \geq \exp(-\text{poly}(\max_i |V_i|)\log(t))$, this equals the size of the maximum independent set. Then a witness is a maximum independent set which is maximum in all connected components, and we can read off whether each $(G_i, \theta_i) \in$ Independent Set and, if so, a witness for this. Since Independent Set is PC-complete, and this is a constructive AND-composition of degree 1, the claim follows from Theorem 3.2 since any instance can be represented with $|V|\log(|V|)(\text{poly}(\texttt{pw}))$ bits.