




On Polynomially Many Queries to NP or QMA Oracles

Sevag Gharibian   

Department of Computer Science and Institute for Photonic Quantum Systems (PhoQS),
Paderborn University, Germany

Dorian Rudolph   

Department of Computer Science and Institute for Photonic Quantum Systems (PhoQS),
Paderborn University, Germany

Abstract

We study the complexity of problems solvable in deterministic polynomial time with access to an NP or Quantum Merlin-Arthur (QMA)-oracle, such as P^{NP} and P^{QMA} , respectively. The former allows one to classify problems more finely than the Polynomial-Time Hierarchy (PH), whereas the latter characterizes physically motivated problems such as Approximate Simulation (APX-SIM) [Ambainis, CCC 2014]. In this area, a central role has been played by the classes $P^{NP[\log]}$ and $P^{QMA[\log]}$, defined identically to P^{NP} and P^{QMA} , except that only *logarithmically* many oracle queries are allowed. Here, [Gottlob, FOCS 1993] showed that if the adaptive queries made by a P^{NP} machine have a “query graph” which is a tree, then this computation can be simulated in $P^{NP[\log]}$.

In this work, we first show that for any verification class $C \in \{NP, MA, QCMA, QMA, QMA(2), NEXP, QMA_{exp}\}$, any P^C machine with a query graph of “separator number” s can be simulated using deterministic time $\exp(s \log n)$ and $s \log n$ queries to a C -oracle. When $s \in O(1)$ (which includes the case of $O(1)$ -treewidth, and thus also of trees), this gives an upper bound of $P^{C[\log]}$, and when $s \in O(\log^k(n))$, this yields bound $QP^{C[\log^{k+1}]}$ (QP meaning quasi-polynomial time). We next show how to combine Gottlob’s “admissible-weighting function” framework with the “flag-qubit” framework of [Watson, Bausch, Gharibian, 2020], obtaining a unified approach for embedding P^C computations directly into APX-SIM instances in a black-box fashion. Finally, we formalize a simple no-go statement about polynomials (c.f. [Krentel, STOC 1986]): Given a multi-linear polynomial p specified via an arithmetic circuit, if one can “weakly compress” p so that its optimal value requires m bits to represent, then P^{NP} can be decided with only m queries to an NP-oracle.

2012 ACM Subject Classification Theory of computation \rightarrow Quantum complexity theory; Theory of computation \rightarrow Complexity classes

Keywords and phrases admissible weighting function, oracle complexity class, quantum complexity theory, Quantum Merlin Arthur (QMA), simulation of local measurement

Digital Object Identifier 10.4230/LIPIcs.ITCS.2022.75

Related Version *Full Version:* [arXiv:2111.02296](https://arxiv.org/abs/2111.02296) [19]

Funding *Sevag Gharibian:* DFG grants 432788384 and 450041824.

Dorian Rudolph: DFG grant 432788384.

Acknowledgements We thank Eric Allender, Johannes Bausch, Stephen Piddock, James Watson, and Justin Yirka for helpful discussions.

1 Introduction

The celebrated Cook-Levin Theorem [13, 35] and Karp’s 21 NP-complete problems [29] laid the groundwork for the theory of NP-completeness to become the *de facto* “standard” for characterizing “hard” problems. Indeed, in the decades since, hundreds of decision problems



© Sevag Gharibian and Dorian Rudolph;

licensed under Creative Commons License CC-BY 4.0

13th Innovations in Theoretical Computer Science Conference (ITCS 2022).

Editor: Mark Braverman; Article No. 75; pp. 75:1–75:27

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

have been identified as NP-complete (see, e.g., [16]). Yet, despite the success of this theory, it soon became apparent that finer characterizations were needed to capture the complexity of certain hard problems.

In this direction, Stockmeyer [42] defined the *Polynomial Hierarchy* (PH), of which the second level will interest us here. Specifically, one may consider $\Sigma_2^P = \text{NP}^{\text{NP}}$ (i.e. an NP-machine with access to an NP-oracle) or $\Delta_2^P = \text{P}^{\text{NP}}$ (i.e. a P machine with access to an NP-oracle). Here, our focus is on the latter, defined as the set of decision problems solvable by a deterministic polynomial-time Turing machine making polynomially many queries to an oracle for (say) SAT. Like NP, P^{NP} has natural complete problems, such as that shown by Krentel [34]: Given Boolean formula $\phi : \{0, 1\}^n \rightarrow \{0, 1\}$, does the lexicographically largest satisfying assignment $x_1 \cdots x_n$ of ϕ have $x_n = 1$?

Restricting the number of NP queries. In 1982, in pursuit of yet finer characterizations, Papadimitriou and Zachos [38] asked: What happens if one considers problems “slightly harder” than NP, i.e. solvable by a P machine making only *logarithmically* many queries to an NP-oracle? This class, denoted $\text{P}^{\text{NP}[\log]}$, contains both NP and co-NP (since the P machine can *postprocess* the answer of the NP-oracle by negating said answer), and is thus believed strictly harder than NP. The following decade saw a flurry of activity on this topic (see Section 1.3); for example, Wagner [43, 44] showed that deciding if the optimal solution to a MAX- k -SAT instance has even Hamming weight is $\text{P}^{\text{NP}[\log]}$ -complete.

This led to the natural question: Is $\text{P}^{\text{NP}[\log]} = \text{P}^{\text{NP}}$? If one restricts the P^{NP} machine to make all NP queries in *parallel* (i.e. non-adaptively), denoted $\text{P}^{\parallel\text{NP}}$, then Hemachandra [27] and Buss and Hay [10] have shown $\text{P}^{\parallel\text{NP}} = \text{P}^{\text{NP}[\log]}$. Thus, adaptivity appears crucial; so, Gottlob [22] next allowed dependence between queries as follows: One may view P^{NP} as a directed acyclic graph (DAG), whose nodes represent NP queries, and directed edge (u, v) indicates that query v depends on the answer of query u . Denote this as the “query graph” of the P^{NP} computation (Definition 23). In 1995, Gottlob showed that any P^{NP} computation whose query graph is a tree can be simulated in $\text{P}^{\text{NP}[\log]}$. To the best of our knowledge, this is the current state of the art regarding P^{NP} versus $\text{P}^{\text{NP}[\log]}$.

Developments on the quantum side. A few years later, the complexity theoretic study of “quantum constraint satisfaction problems” began in 1999 with Kitaev “quantum Cook-Levin theorem” [32], which states that the problem of estimating the “ground state energy” of a local Hamiltonian (k -LH) is complete for Quantum Merlin Arthur (QMA, a quantum generalization of NP). Particularly appealing is the fact that k -LH is physically motivated: It encodes the problem of estimating the energy of a quantum system when cooled to its lowest energy configuration.

More formally, k -LH generalizes the problem MAX- k -SAT, and is specified as follows. As input, we are given a (succinct) description of a Hermitian matrix $H = \sum_i H_i \in \mathbb{C}^{2^n \times 2^n}$, where each Hermitian H_i is a local “quantum clause” acting non-trivially on at most k qubits (out of the full n -qubit system). The *ground state* (i.e. optimal assignment) is then the eigenvector of H with the smallest eigenvalue, which we call the *ground state energy* (i.e. optimal assignment’s value). Thus, understanding the low temperature properties of a many-body system is “simply” an eigenvalue problem for some succinctly described exponentially large matrix H . Since Kitaev’s work, a multitude of other physical problems have been shown to be QMA-complete (see, e.g., surveys [36, 8, 17]).

The formalization of $\text{P}^{\text{QMA}[\log]}$. In 2014, Ambainis tied the study of QMA and $\text{P}^{\text{NP}[\log]}$ together by discovering the first $\text{P}^{\text{QMA}[\log]}$ -complete problem ($\text{P}^{\text{QMA}[\log]}$ is defined as P^{NP} , but with the NP-oracle replaced with a QMA-oracle): *Approximate Simulation* (APX-SIM).

To define APX SIM, suppose we wish to simulate the experiment of cooling down a quantum many-body system, and then performing a local measurement so as to extract information about the ground state’s properties. Formalized (roughly) as a decision problem, we must decide, given Hamiltonian H describing the system, observable A describing a local measurement, and inverse polynomially gapped thresholds α and β , whether there exists a ground state $|\psi\rangle$ of H with expected value $\langle\psi|A|\psi\rangle$ below α .

For context, APX-SIM can be viewed as a quantum variation of Wagner’s $\text{P}^{\text{NP}[\log]}$ -complete problem above [43, 44] (does the optimal solution to a MAX-SAT instance have even Hamming weight?), since both problems ask about properties of optimal solutions to quantum and classical constraint satisfaction problems, respectively. However, in the quantum setting, APX-SIM has the additional perk of being strongly physically motivated. This is because often in practice, one is not interested in the ground state energy, but in properties of the *ground state itself* (e.g. does it exhibit certain quantum phenomena? When does it undergo a phase transition?) [17]. APX-SIM models the “simplest” experiment for computing such ground state properties, making no assumptions about additional information the experimenter might *a priori* have. (For example, in APX-SIM, although the goal is to probe the ground state of H , one is *not* given the corresponding ground state energy as input. This is crucial, both complexity theoretically and physically, since in practice an experimenter does not *a priori* know the ground state energy, as it is QMA-complete to compute to begin with!)

$\text{P}^{\text{QMA}[\log]}$ versus P^{QMA} and this paper. This sets up the question inspiring the current work – is $\text{P}^{\text{QMA}[\log]} = \text{P}^{\text{QMA}}$? In 2020, Gharibian, Piddock, and Yirka [18] showed that $\text{P}^{\text{QMA}[\log]} = \text{P}^{\parallel\text{QMA}}$, for $\text{P}^{\parallel\text{QMA}}$ defined as $\text{P}^{\parallel\text{NP}}$ but with an NP-oracle. This gave a quantum analogue of $\text{P}^{\parallel\text{NP}} = \text{P}^{\text{NP}[\log]}$ [27, 10], although it required completely different proof techniques. In this paper, we thus set our sights on the next step: Gottlob’s work on P^{NP} computations with trees as query graphs [22]. What we are able to achieve is not just a quantum analogue of [22], but a significant strengthening in multiple directions for both NP and QMA: Our main result considers query graphs of *bounded separator number* (which includes bounded treewidth, and hence trees), applies to a host of verification classes including NP and QMA, and gives non-trivial (quasi-polynomial) upper bounds even beyond the bounded separator number case. Along the way, we show how to combine the techniques used with the existing work on APX-SIM and $\text{P}^{\text{QMA}[\log]}$, yielding a unified framework for mapping P^{QMA} -type problems directly to APX-SIM instances.

1.1 Our results

To state our results, define $\mathcal{QV} := \{\text{NP}, \text{MA}, \text{QCMA}, \text{QMA}, \text{QMA}(2), \text{NEXP}, \text{QMA}_{\text{exp}}\}$ and $\mathcal{QV}^+ := \mathcal{QV} \cup \{\text{StoqMA}\}$ (formal definitions in Section 2). This is the set of classical and quantum verification classes for which our results will be stated. However, our framework applies in principle to verification classes C beyond these sets; the main properties we require are for C to allow promise gap amplification and classical preprocessing before verification.

Recall now that an NP query graph is a DAG encoding an arbitrary P^{NP} computation, where nodes correspond to NP queries; denote this an NP-DAG. Replacing NP with any $C \in \mathcal{QV}^+$, we arrive at the notion of a C-DAG (Definition 23). As expected, deciding whether a given C-DAG corresponds to an accepting P^C computation is itself a P^C -complete problem (Lemma 26). To thus obtain new upper bounds on P^C computations, in this work, we parameterize a given C-DAG via its *separator number*, s .

Briefly, a graph $G = (V, E)$ on n vertices has a *separator* of size $s(n)$ if there exists a set of at most $s(n)$ vertices whose removal splits the graph into at least two (non-empty) connected components (Definition 18). G has *separator number* [23] $s(n)$ if, (1) for all subsets $Q \subseteq V$, the vertex-induced graph on Q has a separator of size at most $s(n)$, and (2) $s(n)$ is the smallest number for which this holds. Denote by $C\text{-DAG}_s$ a C -DAG of separator number s , where we write $C\text{-DAG}_1$ for the case of $s \in O(1)$. Note that treewidth upper bounds separator number [23].

1. Deciding C-DAGs. Our main result is the following. For clarity, by “deciding” a C -DAG, we mean deciding whether it encodes an accepting or rejecting P^C computation.

► **Theorem 1.** *Fix any $C \in \mathcal{QV}$ and efficiently computable function $s : \mathbb{N} \rightarrow \mathbb{N}$. Then, $C\text{-DAG}_s \in \text{DTIME}(2^{O(s(n)\log n)})^{C[s(n)\log n]}$, for n the number of nodes in G .*

In words, any P^C computation with a query graph of separator number s can be simulated by a classical deterministic Turing machine running in time $2^{O(s(n)\log n)}$ and making $s(n)\log n$ queries to a C -oracle. With Theorem 1 in hand, we obtain the following sequence of results.

First, we significantly strengthen Gottlob’s [22] $\text{TREES}(\text{NP}) = P^{\text{NP}[\log]}$ result to the constant separator number case ($s = O(1)$) and a broad range of verification classes C :

► **Theorem 2.** *For any $C \in \mathcal{QV}$, $C\text{-DAG}_1$ is $P^{C[\log]}$ -complete.*

In words, any P^C computation with a query graph of separator number $O(1)$ is decidable in $P^{C[\log]}$. Second, an advantage of Theorem 1 is that it scales with *arbitrary* $s(n)$. Thus, to our knowledge, we obtain the first upper bounds for P^C involving *quasi*-polynomial resources:

► **Corollary 3.** *For all integers $k \geq 1$ and $C \in \mathcal{QV}$, $C\text{-DAG}_{\log^k} \in \text{QP}^{C[\log^{k+1}(n)]}$, where QP denotes quasi-polynomial time (Definition 12).*

In words, any P^C computation with a query graph of polylogarithmic separator number is decidable in quasi-poly-time with polylog C -queries. In general, $s(n)$ may scale as $O(n)$, in which case Theorem 1 does not yield a non-trivial bound. Whether this can be improved is left as an open question (Section 1.4).

Third, an example of a verification class which is *not* known to satisfy promise gap amplification is StoqMA (see, e.g., [3]). Here, we also obtain non-trivial bounds, albeit weaker ones:

► **Theorem 4.** *Fix $C = \text{StoqMA}$ and any efficiently computable function $s : \mathbb{N} \rightarrow \mathbb{N}$. Then, $C\text{-DAG}_s \in \text{DTIME}(2^{O(s(n)\log^2 n)})^{C[s(n)\log^2 n]}$.*

Note the extra log factor in the exponents – this prevents Theorem 4 from recovering result $P^{\|\text{StoqMA}} = P^{\|\text{StoqMA}[\log]$ [18] ($P^{\|\text{StoqMA}}$ corresponds to a StoqMA-DAG with $s(n) = 1$). Nevertheless, we *do* recover and improve on [18] when we instead consider the case of bounded *depth* query graphs next.

Finally, Gottlob [22] also studied query graphs of bounded *depth*. The next theorem is an extension of his result. We define $C\text{-DAG}_d$ as $C\text{-DAG}_s$, except now we consider query DAGs of *depth*¹ at most d (as opposed to separator number s).

¹ We define $\text{depth}(G)$ as the maximum distance to any node u from the closest v with $\text{indeg}(v) = 0$.

► **Theorem 5.** *Let $d : \mathbb{N} \rightarrow \mathbb{N}$ be an efficiently computable function. For $C \in \{\text{NP}, \text{NEXP}, \text{QMA}_{\text{exp}}\}$, $C\text{-DAG}_d \subseteq \text{P}^{C[d(n)\log(n)]}$, and for $C \in \mathcal{QV}^+$ it holds that, $C\text{-DAG}_d \subseteq \text{DTIME}(2^{O(d(n)\log(n))})^{C[d(n)\log(n)]}$.*

► **Corollary 6.** *For $C \in \mathcal{QV}^+$ and $d \in O(1)$, $C\text{-DAG}_d$ is $\text{P}^{C[\log]}$ -complete.*

Using this, we obtain that deciding a P^C computation with a query graph of constant depth is $\text{P}^{C[\log]}$ -complete (Corollary 6). This modestly improves upon $\text{P}^{\|\text{StoqMA}\|} = \text{P}^{\text{StoqMA}[\log]}$ [18], which is the case of $d = 1$ (versus our $d \in O(1)$ in Theorem 5).

2. A unified framework for embedding P^C problems into APX-SIM. To date, there are two known approaches for embedding QMA-oracle queries (and thus $\text{P}^{\text{QMA}[\log]}$ problems) into APX-SIM: The “query gadget” construction of Ambainis [5], and the “flag-qubit” framework² of Watson, Bausch, and Gharibian [46]. Each of these frameworks has complementary pros and cons: The former handles *adaptive* oracle queries, but is difficult to use when strong *geometric* constraints for APX-SIM are desired (e.g. the physically motivated settings of 1D and/or translationally invariant Hamiltonians), whereas the latter requires non-adaptive queries, but is essentially agnostic to the circuit-to-Hamiltonian³ mapping used (and thus easily handles geometric constraints).

Here, we utilize the construction behind our main result, Theorem 1, to unify these approaches into a single framework for embedding arbitrary P^C computations into APX-SIM. The crux of the reduction is the following “generalized lifting lemma”, whose full technical statement (Lemma 42) is beyond the scope of this introduction.

► **Lemma 7** ((Informal) Generalized Lifting Lemma (c.f. Lifting Lemma of [46])). *Fix $C \in \mathcal{QV}^+$ and any local circuit-to-Hamiltonian mapping H_w (Definition 41). Define $N_d := 2^{O(d(n)\log n)}$, and $N_s := 2^{O(s(n)\log n)}$ if $C \in \mathcal{QV}$ or $N_s := 2^{O(s(n)\log^2 n)}$ if $C = \text{StoqMA}$. Define $N := \min(N_s, N_d)$, and let G be a $C\text{-DAG}$ instance n vertices of separator number $s(n)$ (as in Theorem 1) and depth $d(n)$ (as in Theorem 5). Then, there exists a $\text{poly}(N)$ -time many-one reduction from G to an instance (H, A) of APX-SIM, such that H has size $\text{poly}(N)$ and satisfies all geometric properties of H_w (e.g. locality of clauses, 1D nearest-neighbor interactions, etc).*

In words, one can embed any P^C computation directly into an APX-SIM instance H in $\text{poly}(N)$ time, irrespective of the choice of C or H_w (i.e. the mapping is essentially black-box). For clarity, a lifting lemma for APX-SIM was first given in [46], which our Lemma 7 generalizes as follows: (1) [46] requires parallel queries to C , whereas Lemma 7 allows arbitrary P^C computations (parameterized by separator number s), and (2) [46] requires promise gap amplification for C , which is not known to hold for StoqMA, whereas Lemma 7 allows $C = \text{StoqMA}$.

Next, by applying our lifting lemma for $C = \text{QMA}$ and $s \in O(1)$, we obtain $\text{P}^{\text{QMA}[\log]}$ -hardness of APX-SIM (Theorem 46). This is not surprising, since Theorem 2 shows $C\text{-DAG} \in \text{P}^{\text{QMA}[\log]}$, and APX-SIM is $\text{P}^{\text{QMA}[\log]}$ -hard [5, 20]. What *is* interesting, however, is:

1. The map from P^C to APX-SIM of Lemma 7 is “direct”, meaning we embed all the query dependencies of the input $C\text{-DAG}$ directly into the flag qubit construction.

² This is a significantly generalized version of the “sifter” construction of Gharibian and Yirka [20].

³ Here, a “circuit-to-Hamiltonian mapping” is a quantum analogue of the Cook-Levin construction, i.e. a map from quantum verification circuits to local Hamiltonians.

2. A poly-time reduction from P^{QMA} to APX-SIM for all $1 \leq s \leq n$ would imply $P^{\text{QMA}} = P^{\text{QMA}[\log]}$ and is therefore unlikely, if one believes $P^{\text{QMA}} \neq P^{\text{QMA}[\log]}$. However, Lemma 42 shows P^{QMA} can be embedded into APX-SIM, at the expense of blowing up the APX-SIM instance's size to $N = 2^{O(s(n) \log n)}$.
3. Finally and most interestingly, the construction of [46] is somewhat mysterious, in that it “compresses” multiple QMA query answers into a *single* flag qubit, which *a priori* appears at odds with Holevo’s theorem. In the present paper, we reveal *why* this works – our construction utilizes the “admissible weighting function” framework of [22], which Gottlob used to reduce P^{NP} computations to maximization of a real-valued function, f . But as we discuss in Section 1.2, this is precisely what the flag qubit framework allows one to simulate (in both [46] and here)!

In fact, we observe that [5] implicitly rediscovers⁴ a version of Gottlob’s weighting function approach. Thus, underlying all three works of [22, 5, 46], as well as the current one, is a central unifying theme worth stressing:

► **Theme 8** (Unifying theme). *The reduction of P^C to maximizing a real-valued function.*

Finally, for $C = \text{StoqMA}$ and $s \in O(1)$, application of our lifting lemma is still possible (i.e. utilizing the N_s term), but the Hamiltonian obtained is now quasi-polynomial in size, since $N := 2^{O(s(n) \log^2 n)}$ (Theorem 47). Luckily, we can instead utilize the N_d term (i.e. bounded-depth setup) of the lifting lemma, which yields the desired poly(n)-size output Hamiltonian when $d \in O(1)$. This means we recover the $P^{\text{StoqMA}[\log]}$ -hardness result of [18] via the flag qubit framework (details in Section 5.2) resolving an open question of [46]. For clarity, [18]’s proof of this result is via perturbation theory, which we do not require here.

3. No-go statement for “weak compression” of polynomials. To further drive home the point of Theme 8, we close with a simple no-go statement purely about polynomials. Roughly, given a real-valued polynomial f (specified via an arithmetic circuit), we define *weak compression* as efficiently mapping f to an efficiently computable real-valued function g , such that there exists an optimal point y^* at which g is maximized, from which (1) we may efficiently recover an optimal point x^* maximizing f , and (2) $g(y^*)$ requires fewer bits than $f(x^*)$ to represent (i.e. has been “compressed”).

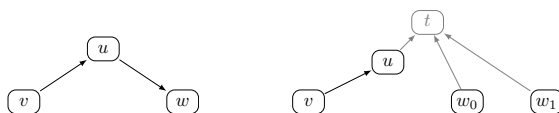
► **Lemma 9.** *Fix any function $h : R^+ \rightarrow R^+$. Suppose that given any multi-linear polynomial p (represented as an arithmetic circuit) requiring B bits for some optimal solution (in the sense of Definition 48), p is weakly compressible to $h(B)$ bits. Then $P^{\text{NP}} \subseteq P^{\text{NP}[h(B)]}$.*

Let us be clear that this statement is not at all surprising for the reader familiar with Krentel’s work [33] on OptP (see Section 1.3). Nevertheless, we believe it is worth formalizing, as it uses complexity theory to give a no-go statement about a purely mathematical concept (non-compressibility of polynomials). From Lemma 9, one obtains:

► **Corollary 10.** *If any multi-linear polynomial p (represented as an arithmetic circuit) can be weakly compressed with $h(B) = O(\log B)$, then $P^{\text{NP}} \subseteq P^{\text{NP}[\log]}$.*

► **Corollary 11.** *If any multi-linear polynomial p requiring $B \in O(1)$ bits for some optimal solution can be weakly compressed with $h(B) = 1$, then the Polynomial-Time Hierarchy (PH) collapses to its third level (more accurately, to $P^{\Sigma_2^p}$).*

⁴ Like [22], [5] uses an exponentially growing weighting function to ensure soundness when simulating adaptive oracle queries, although the term “admissible weighting function” is not used in the latter.



■ **Figure 1** Simple example of a graph transformation, where the outputs of u are decoupled by creating copies w_0, w_1 with hardcoded inputs. t selects the copy of w depending on the output of u .

1.2 Techniques

1. Techniques for deciding C-DAGs. At a high-level, our approach follows Gottlob’s strategy for P^{NP} [22]: Given a C-DAG G , we (1) “compress” G to an equivalent query G' , (2) define an “admissible weighting function” on G' , (3) define an appropriate verifier V , on which binary search via C-oracle queries suffices to extract the original C-query answers in G , and thus to decide G itself. The key steps at which we deviate significantly from [22] are (1) and (3), as we now elaborate.

In more detail, in order to decide G , the goal is to compute a *correct query string* x for G , i.e. a string of answers to the C-oracle queries asked by G . (Note x is not necessarily unique when C is a *promise* class such as QMA.) For this, fix any topological order T on the nodes of G . The clever insight of [22] (rediscovered in [5]), is that by “weighting” queries early in T exponentially larger than queries later in T , one can force all queries, in order, to be answered correctly. Roughly speaking, such an exponential weighting scheme ω is called “admissible” (Definition 30). The core premise is then to map (G, ω) to a real-valued function f , so the maximum value of f encodes the query string x . Hence, by conducting binary search on f via the C-oracle, one can identify f ’s optimal value, thus recovering x . The challenge is that for *arbitrary* G , the maximum value of f can scale exponentially in n , the number of nodes in G . Thus, one requires $\text{poly}(n)$ queries to extract x , obtaining no improvement over the P^C computation G we started with!

Compressing G . To overcome this in our setting of bounded separator number (and beyond), we first recursively compute separators of G , obtaining a “separator tree” (Definition 20) structure overlaying G . With this separator tree in hand, we show our main technical lemma, the Compression Lemma (Lemma 32). Roughly, the idea behind the Compression Lemma is to “decouple” dependencies in G by creating multiple copies of a node. To illustrate, an oversimplified example is given in Figure 1, where the output node w depends on u , which depends on v . (Each node encodes, say, an NP query.) To remove the dependency of w on u , we create two copies w_0 and w_1 , where the input from u is hardcoded as 0 or 1, respectively. Then an output node t is added to select the correct copy of w depending on the output of v .

For clarity, this basic decoupling principle is reminiscent of that employed in [22]. However, whereas the latter maps G to G' via iterative local transformations (similar to Figure 1, but without the t node), here we are unable to make such an approach work. Indeed, due to the much stronger coupling between nodes in our setting, we appear to acquire a carefully orchestrated, *global* transformation of G to G' . Roughly, we must carefully exploit the separator tree as a guide to recursively create node copies and reroute wires, at the end of which we introduce a “conductor” node t to orchestrate the madness. For the reader interested in a brief peek at details (Section 4.2), Figure 2 runs through an example graphically depicting the global compression, and Algorithm 2 is used (e.g.) in t to recursively orchestrate and compute the final output of the new C-DAG, G' . The upshot of this global transformation is that, when $s \in O(1)$, G' is “compressed” in such a way that (roughly) we can define an admissible weighting function of at most $\text{poly}(n)$ weight on G' , as we do next.

Designing the verifier V . The second main step (Section 4.3) is to use an admissible weighting function on G' to “reduce” G' to maximization of a real-valued function, t (Theme 8); we use (Equation (4))

$$t(x, \psi_1, \dots, \psi_N) := \sum_{i=1}^N f(v_i) (x_i \Pr[Q_i(z_i(x), \psi_i) = 1] + (1 - x_i)\gamma), \quad (1)$$

where intuitively, $f(v_i)$ is the weight at node i , and $\Pr[Q_i(z_i(x), \psi_i) = 1]$ is the probability that C -verifier Q_i at node v_i accepts, given incoming wires $z_i(x)$ from its parents and proof $|\psi_i\rangle$. Function t is carefully designed so that (1) any “approximately maximum” value of t encodes a correct query string x (Lemma 39), and (2) we can design a C -verifier V with acceptance probability precisely $t(x, \psi_1, \dots, \psi_N)$ (up to renormalization) (Lemma 38). Thus, binary search via V allows us to extract x from t . Crucially, by the compression of the previous step, when $s \in O(1)$, the maximum value of t is at most $\text{poly}(n)$, meaning $O(\log n)$ C -queries suffice in the binary search. Moreover, our V is simple – it simulates a random Q_i (according to the distribution induced by weights $f(v_i)$) on $(x, |\psi_i\rangle)$. We exploit this by defining t over a *cross product* of proofs $|\psi_i\rangle$ (rather than a tensor product, as is usual), avoiding complications regarding entanglement across proofs from previous works (e.g. [46]).

2. Techniques for a unified APX-SIM framework. Roughly, [46] embeds a (say) $\text{PQMA}^{\text{[log]}}$ computation Π into APX-SIM as follows: (1) Build a “superverifier” circuit V , which verifies each of the queries of Π in parallel, and conditioned on the output of each subverifier, performs a small rotation on a shared “flag qubit”, q . The superverifier V is then pushed through an abstract circuit-to-Hamiltonian mapping H_w , and the encoding of q in the resulting Hamiltonian $H_w(V)$ is carefully penalized to force low energy states to correctly answer all queries. The advantage of this setup is that it is oblivious to the choice of H_w ; the disadvantage is that it requires a somewhat involved exchange argument to ensure soundness against entanglement across parallel proofs.

Recall now that our main construction rolls up an entire arbitrary C -DAG into a single C -verifier, V (Lemma 38). What we next show is that our V can rather simply be substituted for the superverifier V of [46] in the flag qubit construction. The key reason this works is again Theme 8 – since, as mentioned above, the acceptance probability of our V literally encodes the value of t , we can treat the output wire of our V as the “new flag qubit” q (thus eliminating the multiple rounds of small rotations in [46]). As in [46], by then mapping V to $H_w(V)$, we can now penalize q on the Hamiltonian side to force all low energy states of $H_w(V)$ to implicitly maximize t ! Finally, we remark that since our V is naturally robust against entanglement across proofs, our proof of correctness is significantly simpler than [46].

3. Techniques for “weak compression” of polynomials. This result follows easily by combining Section 4.3.2 with standard techniques, so we keep the discussion brief. Roughly, given an NP-DAG, we (1) apply the Cook-Levin theorem to map each NP verifier to a SAT formula, (2) arithmetize each of these SAT formula and combine them to simulate Equation (1) on the Boolean hypercube, and (3) linearize the resulting multi-variate polynomial; denote the output as p . Since p is multilinear, it is maximized on our domain of interest on a vertex of the hypercube; thus, by design, from the maximum value of p , we can recover the maximum value of t , from which we can extract the correct query string for the input NP-DAG. The argument is concluded by observing that to identify the maximum p^* of p , a binary search via NP-oracle requires $O(\log(p^*))$ queries. As an aside, the collapse of PH in Corollary 11 leverages Hartmanis’ result that if $\text{P}^{\text{NP}[2]} = \text{P}^{\text{NP}[1]}$, then $\text{PH} = \text{P}^{\Sigma_2^p}$ [26].

1.3 Related Work

The classes \mathbf{P}^{NP} and $\mathbf{P}^{\text{NP}[\log]}$. As mentioned above, $\text{NP} \cup \text{coNP} \subseteq \mathbf{P}^{\text{NP}[\log]} \subseteq \Sigma_2^p$, and $\mathbf{P}^{\text{NP}[\log]} \subseteq \text{PP}$ [6]. It holds that $\mathbf{P}^{\text{NP}[\log]} = \text{P}^{\parallel \text{NP}}$ [27, 10]. Gottlob [22] showed that \mathbf{P}^{NP} with a tree for a query graph equals $\mathbf{P}^{\text{NP}[\log]}$ (this also follows from our Theorem 2). It is believed that for any $k \in O(1)$, the classes $\mathbf{P}^{\text{NP}[k]}$, $\mathbf{P}^{\text{NP}[\log^k n]}$, and \mathbf{P}^{NP} are distinct. For example, $\mathbf{P}^{\text{NP}[1]} = \mathbf{P}^{\text{NP}[2]}$ implies both $\mathbf{P}^{\text{NP}[1]} = \mathbf{P}^{\text{NP}[\log]}$ and a collapse of PH to $\Delta_3^p = \text{P}^{\Sigma_2^p}$ [26]. However, it is known that $\mathbf{P}^{\text{NP}[\log^k(n)]} = \text{P}^{\parallel \text{NP}[\log^{k+1}(n)]}$ for all $k \geq 1$ [11]. Complete problems for $\mathbf{P}^{\text{NP}[\log]}$ include determining a winner in Lewis Carroll’s 1876 voting system [28], and a $\mathbf{P}^{\text{NP}[\log^2 n]}$ -complete problem is model checking for certain branching-time temporal logics [41].

Closely related to one of the central themes of this work, Theme 8, is Krentel’s [33] work on OptP. Roughly, $\text{OptP}[z(n)]$ is the class of *functions* (i.e. not decision problems) computable via maximization of a real-valued function, where the function is restricted to $z(n)$ bits of output precision. Krentel shows the classes $\text{OptP}[z(n)]$ and $\text{FP}^{\text{NP}[z(n)]}$ are equivalent (FP the set of *functions* computable in poly-time). Through this, [33] obtains (e.g.) that determining whether the length of the shortest traveling salesperson tour in a graph G is divisible by k is \mathbf{P}^{NP} -complete, but that determining if the size of the max clique in G is divisible by k is only $\mathbf{P}^{\text{NP}[\log]}$ -complete. Before this, Papadimitriou had shown [37] that deciding if G has a *unique* optimum traveling salesperson tour is \mathbf{P}^{NP} -complete.

QMA, $\mathbf{P}^{\text{QMA}[\log]}$ and related classes. Kitaev’s “quantum Cook-Levin/circuit-to-Hamiltonian” construction showing QMA-completeness for the local Hamiltonian problem has since been greatly extended to many settings (e.g. [31, 30, 2, 21]). For $\text{QMA}(2)$, Chailloux and Sattath [12] showed the *separable sparse Hamiltonian problem* is $\text{QMA}(2)$ -complete. Fefferman and Lin [15] prove that the local Hamiltonian problem with *exponentially* small promise gap is PSPACE-complete. See (e.g.) [36, 17] for surveys and further results.

Ambainis [5] initiated the study of $\mathbf{P}^{\text{QMA}[\log]}$, and showed APX-SIM is $\mathbf{P}^{\text{QMA}[\log]}$ -complete and SPECTRAL GAP (deciding if the spectral gap of a local Hamiltonian is large or small) is $\text{P}^{\text{UQMA}[\log]}$ -hard. These results were obtained for log-local observables (APX-SIM) and Hamiltonians (APX-SIM and SPECTRAL GAP). Gharibian and Yirka [20] improve both results to $O(1)$ -local, and show $\mathbf{P}^{\text{QMA}[\log]} \subseteq \text{PP}$. In contrast to $\mathbf{P}^{\text{NP}[\log]}$, $\mathbf{P}^{\text{QMA}[\log]}$ is *not* believed to be in PH, since even BQP is believed outside of PH [1, 39]. Gharibian, Piddock, and Yirka [18] next obtain a complexity classification of $\mathbf{P}^{\text{QMA}[\log]}$ (along the lines of Cubitt and Montanaro [14]) depending on the class of Hamiltonians employed; this includes, for example, $\text{P}^{\text{StoqQMA}[\log]}$ -completeness for APX-SIM on stoquastic Hamiltonians. They also introduce the “sifter” framework to show the first $\mathbf{P}^{\text{QMA}[\log]}$ -hardness result for 1D Hamiltonians on the line. Watson, Bausch, and Gharibian [46] significantly extend the sifter framework to develop the flag-qubit framework (also used in Section 5), showing (among other results) that APX-SIM on 1D translation-invariant systems is $\mathbf{P}^{\text{QMA}_{\text{exp}}}$ -complete.

Most recently, Watson and Bausch [45] show a $\mathbf{P}^{\text{QMA}_{\text{exp}}}$ -completeness result for approximating a critical boundary in the phase diagram of a translationally-invariant Hamiltonian. Aharonov and Irani [4] and Watson and Cubitt [47] simultaneously and independently study variants of the problem of computing digits of the ground state energy of a translationally invariant Hamiltonian in the thermodynamic limit. The former shows that the function version of this problem lies between FP^{NEXP} and $\text{FP}^{\text{QMA}_{\text{exp}}}$, while the latter shows that a decision version of the energy density problem is between P^{NEXP} and $\text{EXP}^{\text{QMA}_{\text{exp}}}$ (for quantum Hamiltonians).

1.4 Open questions

First, can our main result (Theorem 1) be extended to further classes of graphs, perhaps by considering different parameterizations, such as graphs with logarithmic pathwidth? Second, Theorem 1 gives non-trivial bounds when (say) $s \in O(1)$ or $s \in O(\text{polylog}(n))$. For $s \in \Theta(n)$, however, the DTIME base therein scales as 2^n , thus yielding a trivial upper bound on $C\text{-DAG}_s$. Can our bound be improved from $\text{DTIME}(2^{O(s(n)\log n)})^{C[s(n)\log n]}$ to $\text{DTIME}(2^{O(s(n))})^{C[s(n)\log n]}$ (i.e. shave off the extra log factor in the base)? If so, one would immediately recover the $\text{P}^{\text{StoqMA}} \subseteq \text{P}^{\text{StoqMA}[\log]}$ result of [18] (currently, we rely on Theorem 5 to recover this here), and more generally, our framework would not take a hit when applied to classes C without promise gap amplification. However, what is *unlikely* is to show a bound of $\text{DTIME}(2^{O(s(n))})^{C[s(n)]}$ – since P^{NP} has $s \in O(1)$, this would imply $\text{P}^{\text{NP}} = \text{P}^{\text{NP}[\log]} \in \text{P}^{\text{NP}[k]}$ for $k \in O(1)$. Third, do our theorems also hold for complexity classes such as UniqueQMA (UQMA) or QMA_1 (QMA with perfect completeness)? Here, the main difficulty seems to be *invalid* queries (queries violating the promise), as then the verifier from Lemma 38 does not necessarily have a unique proof or perfect completeness. One could also consider AM-like complexity classes instead of the MA-like classes we used.

2 Preliminaries

► **Definition 12** (QP (quasi-polynomial time)). $\text{QP} = \bigcup_k \text{DTIME}(n^{\log^k n})$ is the set of problems accepted by a deterministic Turing machine in quasi-polynomial time.

Quantum Complexity Classes. Qubits are represented by the Hilbert space $\mathcal{B} := \mathbb{C}^2$, where “:=” denotes a definition.

► **Definition 13** (QMA). Fix polynomials $p(n)$ and $q(n)$. A promise problem Π is in QMA (Quantum Merlin Arthur) if there exists a polynomial-time uniform quantum circuit family $\{Q_n\}$ such that the following holds:

- For all n , $Q_n \in \mathcal{U}(\mathcal{B}_A^{\otimes n} \otimes \mathcal{B}_B^{\otimes p(n)} \otimes \mathcal{B}_C^{\otimes q(n)})$. The register A is used for the input, B contains the proof, and C the ancillae initialized to $|0\rangle$.
- $\forall x \in \Pi_{\text{yes}} \exists |\psi\rangle \in \mathcal{B}^{\otimes p(|x|)} : \Pr[Q_{|x|} \text{ accepts } |x\rangle|\psi\rangle] \geq 2/3$
- $\forall x \in \Pi_{\text{no}} \forall |\psi\rangle \in \mathcal{B}^{\otimes p(|x|)} : \Pr[Q_{|x|} \text{ accepts } |x\rangle|\psi\rangle] \leq 1/3$

Note that the thresholds $c = 2/3$ and $s = 1/3$ may be replaced with $c = 1 - \varepsilon$ and $s = \varepsilon$ such that $\varepsilon \geq 2^{-\text{poly}(n)}$ [32]. We refer to c as *completeness*, s as *soundness*, and $c - s$ as the *promise gap*.

We also consider special cases of QMA. In QCMA, the proof is classical, i.e. $|\psi\rangle \in \{0, 1\}^{p(n)}$. In $\text{QMA}(k)$, the verifier receives k unentangled proofs, i.e. $|\psi\rangle = \bigotimes_{j=1}^k |\psi_j\rangle$. It holds that $\text{QMA}(2) = \text{QMA}(\text{poly}(n))$ as shown by Harrow and Montanaro [25]. Therefore, probability amplification is possible. In QMA_{exp} , $p(n)$ and $q(n)$ are allowed to be exponential (i.e. $2^{\text{poly}(n)}$) and $\{Q_n\}$ is an exponential-time uniform quantum circuit family. QMA_{exp} can be considered the quantum analogue of NEXP. StoqMA [9] is a restricted variant of QMA with a classical verifier (details in the full version) and the property that probability amplification is not known to be possible [3].

Next, we define the k -local Hamiltonian problem, which was shown to be QMA-complete in a “quantum Cook-Levin theorem” by Kitaev [32].

► **Definition 14** (*k*-local Hamiltonian). A Hermitian operator $H \in \text{Herm}(\mathcal{B}^{\otimes n})$ acting on n qubits is a *k*-local Hamiltonian if it can be written as $H = \sum_{S \subseteq [n], |S| \leq k} H_S \otimes I_{[n] \setminus S}$. Additionally, $0 \preceq H_S \preceq I$ holds without loss of generality.

We refer to the minimum eigenvalue $\lambda_{\min}(H)$ as the ground state energy of H and the corresponding eigenvectors as ground states.

► **Definition 15** (*k*-LH(H, k, a, b)). Given a *k*-local Hamiltonian $H = \sum_i H_i$ acting on N qubits and real numbers a, b such that $b - a \geq N^{-c}$, for $c > 0$ constant, decide:

- YES. If $\lambda_{\min}(H) \leq a$ (i.e. the ground state energy of H is at most a).
- NO. If $\lambda_{\min}(H) \geq b$.

► **Definition 16** (P^C). Let C be a complexity class with complete problem Π . $P^C = P^\Pi$ is the class of (promise) problems that can be decided by a polynomial-time deterministic Turing machine M with the ability to query an oracle for Π . If M asks an invalid query $x \in \Pi_{\text{inv}}$, the oracle may respond arbitrarily.

- We say $\Gamma \in P^C$ if there exists an M as above such that M accepts/rejects for $x \in \Gamma_{\text{yes}}/x \in \Gamma_{\text{no}}$, regardless of how invalid queries are answered.
- For a function f , we define $P^{C[f]}$ in the same way, but with the restriction that M may ask at most $O(f(n))$ queries on input of length n .
- For an integer k , we define $P^{C[k]}$, where M may ask at most k queries on each input.
- $P^{\parallel C}$ denotes the class where M must ask all queries at the same time. We call these queries non-adaptive opposed to the adaptive queries of the above classes, because the queries do not depend on the results of other queries.

The $P^{\text{QMA}[\log]}$ -complete problem is APX-SIM (approximate simulation). It essentially asks whether a given Hamiltonian has a ground state with a certain property.

► **Definition 17** (APX-SIM(H, A, k, l, a, b, δ)) [5]). Given a *k*-local Hamiltonian $H = \sum_i H_i$ acting on N qubits, an *l*-local observable A , and real numbers a, b , and δ such that $b - a \geq N^{-c}$ and $\delta \geq N^{-c'}$, for $c, c' > 0$ constant, decide:

- YES. If H has a ground state $|\psi\rangle$ satisfying $\langle \psi | A | \psi \rangle \leq a$.
- NO. If for all $|\psi\rangle$ satisfying $\langle \psi | H | \psi \rangle \leq \lambda_{\min}(H) + \delta$, it holds that $\langle \psi | A | \psi \rangle \geq b$.

Graph Theory. Let $G = (V, E)$ be a directed graph. For a node $v \in V$, we define $\text{indeg}(v)$ and $\text{outdeg}(v)$ as the number of incoming and outgoing edges, respectively. The sets $\text{parents}(v) := \{w \in V \mid (w, v) \in E\}$ and $\text{children}(v) := \{w \in V \mid (v, w) \in E\}$ denote the parents and children of v , respectively. The set of *ancestors* (*descendants*) of node v is the set of all $u \in V \setminus \{v\}$, such that there is a directed path in G from u to v (v to u). If G contains no directed cycles, we call it a DAG (directed acyclic graph).

► **Definition 18** (Separator number [23]). Let $G = (V, E)$ be an undirected graph. A set $S \subseteq V$ is a separator of G if $G \setminus S$ (i.e. the graph induced by the nodes $V \setminus S$) has at least two connected components or at most one node. S is balanced if every connected component of $G \setminus S$ has at most $\lceil (|V| - |S|)/2 \rceil$ nodes. The balanced separator number of G , denoted $s(G)$, is the smallest k such that for every $Q \subseteq V$, the induced subgraph $G[Q]$ has a balanced separator of size at most k .

► **Lemma 19** (Theorem 9 of [23] (see also [40, 7])). $s(G) \leq \text{tw}(G) \leq O(s(G) \cdot \log n)$, where $\text{tw}(G)$ denotes the treewidth (minimum width of tree decomposition) of G .

For directed G , $s(G)$ and $\text{tw}(G)$ are defined on the undirected version of G .

Separator Trees. The separator number allows us to decompose graphs into *separator trees*, which we use to evaluate query graphs more efficiently.

► **Definition 20.** A (balanced) separator tree of an undirected graph $G = (V, E)$ is a tree $T = (V_T, E_T)$, with vertices in V_T labelled by disjoint subsets $\{S_1, \dots, S_m\}$ satisfying $\bigcup_{i=1}^m S_i = V$, and T being rooted in S_1 . S_1 is a (balanced) separator of G , and the trees rooted in the children of S_1 are (balanced) separator trees of $G \setminus S_1$. We refer to the vertices/edges of T as supervertices/superedges. A path along superedges is called a superpath. The unique superpath from S_1 to any supervertex S is called a branch of the tree.

Unless noted otherwise, throughout this work we assume separators are balanced. A separator tree can be computed by a straightforward brute force search (see full version for proof):

► **Lemma 21.** Given an n -vertex graph $G = (V, E)$, a separator tree T of G with separator number $s := s(G)$ can be computed in time $n^{O(s)}$.

Without loss of generality, we assume the separator tree computed in Lemma 21 has only separators of size s . Additionally, although a balanced separator tree has $O(\log n)$ depth, at times we may wish to leverage a shorter depth tree if one should exist. For convenience, we hence state the following lemma (proof analogous to Lemma 21).

► **Lemma 22.** Given an n -vertex graph G , depth D and separator size s , a separator tree of G of depth D with separators of size s can be computed in time $n^{O(Ds)}$, if it exists.

3 Query graphs and C -DAG

The main object of study in this work is the concept of a *query graph*, which we now formally define in the context of a decision problem, C -DAG.

► **Definition 23** (C -DAG). Fix any complexity class $C \in \mathcal{QV}^+$. A C -DAG instance is defined by an n -node DAG $G = (V = \{v_1, \dots, v_n\}, E)$, with structure as follows:

- Vertex $v_n \in V$ is the unique vertex with $\text{outdeg}(v_n) = 0$, denoted the result node.
- Each $v_i \in V$ is associated with a promise problem $\Pi^i \in C$ that determines the output of v_i . Formally, Π^i is specified via a $\text{poly}(n)$ -sized description⁵ of a verification circuit Q_i with designated input and proof registers \mathcal{X}_i and \mathcal{Y}_i .⁶ The input register \mathcal{X}_i consists of precisely $\text{indeg}(v_i)$ bits/qubits, set to the string on v_i 's incoming edges/wires. In order to allow non-trivial Q_i for bounded in-degree, we allow an implicit padding of \mathcal{X}_i to $\text{poly}(n)$ bits. v_i has a single output wire, denoted $\text{out-wire}[v_i]$, corresponding to the output of the verifier Q_i .

Finally, we say $G \in C\text{-DAG}_{\text{yes}}$ (respectively, $G \in C\text{-DAG}_{\text{no}}$) if the evaluation procedure EVALUATE (Algorithm 1) outputs 1 (respectively, 0) deterministically (i.e. regardless of how any invalid queries are answered).

► **Definition 24** (Correct query string). Any string $x \in \{0, 1\}^n$ that can be produced via Line 6 of Algorithm 1 is called a correct query string.

► **Remark 25.** Intuitively, in Definition 24 the bits of x encode a sequence of correct query answers corresponding to the nodes of G . Note the correct query string need not be unique if C is a promise class (i.e. invalid queries are allowed).

⁵ This description may be implicit to describe exponentially large circuits (e.g., for NEXP).

⁶ For example, if $C = \text{NP}$, then Π_{yes}^i is the set of all strings x on \mathcal{X}_i , for which there exists a proof y on \mathcal{Y}_i , such that NP verifier Q_i accepts (x, y) .

■ **Algorithm 1** Evaluation procedure for C -DAG.

```

1: function EVALUATE( $G = (V, E)$ )
2:   Sort the nodes of  $V$  topologically into  $v_1, \dots, v_n$ .
3:   The variable  $x_i \in \{0, 1\}$  will denote the result of  $v_i$ 's query.
4:   for  $i = 1, \dots, n$  do
5:      $z_i \leftarrow \bigcirc_{v_j \in \text{parents}(v_i)} x_j$  ▷ “ $\bigcirc$ ” denotes concatenation.
6:      $x_i \leftarrow \begin{cases} 1, & \text{if } z_i \in \Pi_{\text{yes}}^i \\ 0, & \text{if } z_i \in \Pi_{\text{no}}^i \\ 0 \text{ or } 1 \text{ (nondeterministically),} & \text{if } z_i \in \Pi_{\text{inv}}^i \end{cases}$ 
7:   return  $x_n$  ▷ Recall  $v_n$  is the result node.

```

Just as Gottlob shows DAGS(NP) (more accurately, DAGS(SAT)) is P^{NP} -complete [22], here we have the more general statement (proof analogous to [22]):

► **Lemma 26.** *For any $C \in \mathcal{QV}^+$, C -DAG is P^C -complete.*

Thus, Lemma 26 says that on general query graphs G , C -DAG captures all of P^C . The primary aim of this paper is hence to consider graphs G with *bounded separator number* (which, by Lemma 19, includes the case of bounded treewidth). For this, we introduce the following definition for convenience.

► **Definition 27** (C -DAG $_s$). *Let $s : \mathbb{N} \rightarrow \mathbb{N}$ be an efficiently computable function. Then, C -DAG $_s$ is defined as C -DAG, except that G has separator number $s(G) \in O(s(n))$, for n the number of nodes used to specify the C -DAG instance. For brevity, we use C -DAG $_1$ to denote the case of $s \in O(1)$.*

Thus, the union of C -DAG $_s$ over all polynomials $s : \mathbb{N} \rightarrow \mathbb{N}$ equals C -DAG.

4 Query Graphs with Bounded Separator Number

We first state the main technical theorem of this section, Theorem 28, followed by the results we obtain from it as corollaries. The remainder of Section 4 then proves Theorem 28. For clarity, throughout this work, we assume that the full specification of any C -DAG instance G (i.e. the DAG itself, the verification circuits Q_i , etc) scales polynomially with its number of nodes, n . Throughout, any omitted proofs are in the full version.

► **Theorem 28.** *Fix $C \in \mathcal{QV}$. As input, we are given (1) a C -DAG instance G on n nodes, and (2) a separator tree for G of depth D and separator size s . Then, G can be decided in deterministic time $2^{O(sD + \log n)}$ with $O(sD + \log n)$ queries to a C -oracle.*

► **Remark 29.** The class StoqMA is not included in Theorem 28; this is because the proof of the theorem requires C with a constant promise gap, which StoqMA is not known to have. Combining Theorem 28 with Lemma 21 gives:

► **Theorem 1.** *Fix any $C \in \mathcal{QV}$ and efficiently computable function $s : \mathbb{N} \rightarrow \mathbb{N}$. Then, C -DAG $_s \in \text{DTIME}(2^{O(s(n) \log n)})^{C[s(n) \log n]}$, for n the number of nodes in G .*

► **Theorem 2.** *For any $C \in \mathcal{QV}$, C -DAG $_1$ is $P^{C[\log]}$ -complete.*

We obtain the following general scaling corollary for polylogarithmic separator number.

► **Corollary 3.** *For all integers $k \geq 1$ and $C \in \mathcal{QV}$, C -DAG $_{\log^k} \in \text{QP}^{C[\log^{k+1}(n)]}$, where QP denotes quasi-polynomial time (Definition 12).*

Notation. $\Gamma(v) := \{w \mid (v, w) \in E\}$ is the neighbor set of vertex v . The descendants of vertex v are denoted $\text{Desc}(v)$, i.e. the set of nodes reachable from vertex v via a directed path, excluding v itself. Analogously, the ancestors of vertex v are denoted $\text{Anc}(v)$, i.e. the set of nodes from which there is a directed path to v , excluding v itself.

4.1 Weighting Functions

We now introduce the concept of *weighting functions*, which assign a weight to each node in a DAG G . Weighting functions were first used by Gottlob [22] to prove $\text{TREES}(\text{NP}) = \text{P}^{\text{NP}[\log]}$, and later implicitly by Ambainis [5] to show $\text{P}^{\text{QMA}[\log]}$ -hardness of the Approximate Simulation (APX-SIM) problem. We use a modified definition.

► **Definition 30** (Weighting function). *Let $G = (V, E)$ be a DAG. An efficiently computable function $f : V \rightarrow \mathbb{R}$ is called a weighting function. We say f is c -admissible for constant $c \in \mathbb{R}$ if for all $v \in V$, $f(v) \geq 1 + c \sum_{w \in \Gamma(v)} f(w)$, where $\Gamma(v) := \{w \mid (v, w) \in E\}$ is the (out-going) neighbor set of v . The total weight $W_f(G)$ of G under weighting function f is $W_f(G) = \sum_{v \in V} f(v)$.*

In the next lemma, we extend Gottlob’s [22] admissible weighting functions to our definition of c -admissibility (Definition 30). For $c = 1$, the definitions are the same.

► **Lemma 31.** *For any DAG $G = (V, E)$ and $c \geq 2$, the weighting functions $\rho(v) := (c|V|)^{\text{depth}(G) - \text{level}(v)}$ and $\omega(v) := (c + 1)^{|\text{Desc}(v)|}$ are c -admissible.*

In Sections 4.2 and 4.3, we assume $C \in \mathcal{QV}^+$, unless stated otherwise.

4.2 Graph transformation: The Compression Lemma

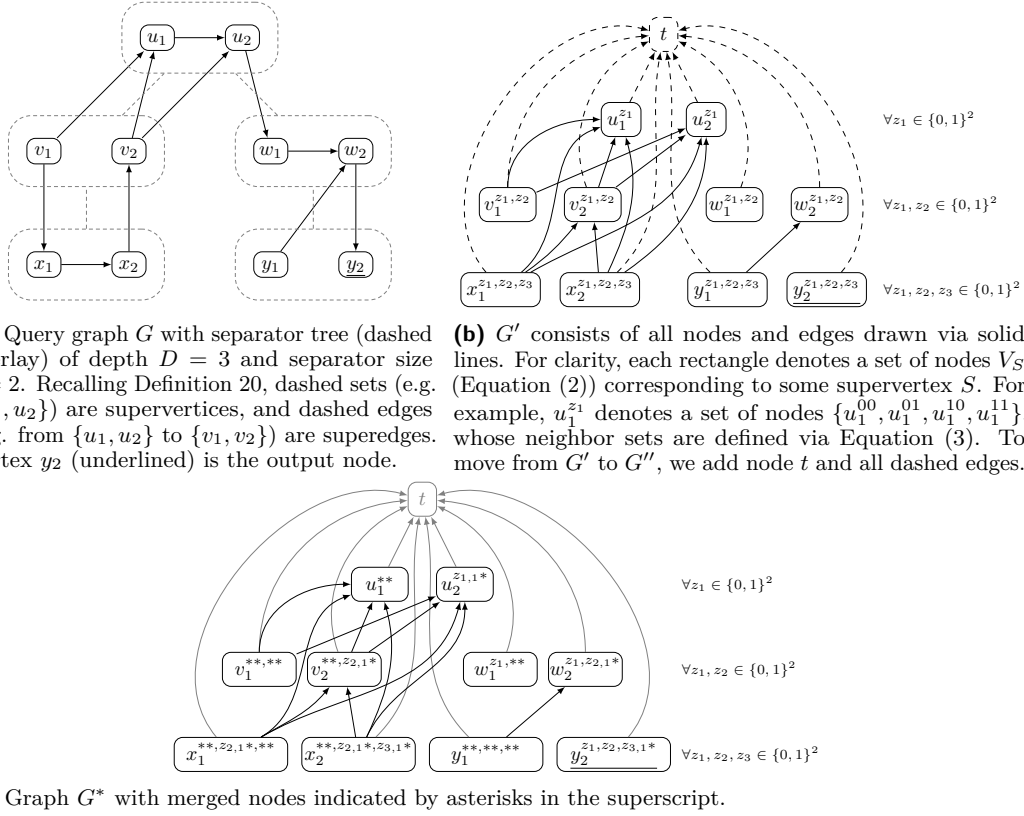
Ideally, our aim for a given C -DAG instance G is to define a c -admissible weighting function f with $W_f(G)$ as small as possible. This is because in Section 4.3, we show how to solve arbitrary C -DAG-instances using $O(\log W_f(G))$ C -queries. Unfortunately, for an arbitrary C -DAG-instance G there does not necessarily exist a c -admissible weighting function f such that $W_f(G)$ is “small”, e.g. subexponential. Thus, in this section, we show:

► **Lemma 32.** *As input, we are given a C -DAG instance G , and a separator tree for G of depth D and separator size s . Fix any constant $c \geq 2$. Then, a query graph $G^* = (V^*, E^*)$ with $|V^*| \leq 2^{O(sD)}n$, together with a c -admissible weighting function f^* and $W_{f^*}(G^*) \leq (c + 1)^{O(sD)}n$, can be constructed in time $2^{O(sD + \log n)}$ such that $\text{EVALUATE}(G) = \text{EVALUATE}(G^*)$. As required by the definition of C -DAG (Definition 23), each node of G^* corresponds to a verification circuit of size $\text{poly}(|V^*|)$.*

Combining this with Section 4.3, we will hence be able to decide G^* with $O(sD)$ queries. In the following we describe the transformation from G to G^* in multiple steps.

4.2.1 Basic Construction (G' and G'')

The graph transformation. Let $T = (V_T, E_T)$ be a separator tree (Definition 20) of G of depth D and separator size s . A running example is given in Figure 2a. Let $S \in V_T$ be an arbitrary supervertex and S_1, \dots, S_d be the unique path along superedges from the root supervertex S_1 to $S_d := S$ (define $d := d_S \leq D$ as the distance from the root plus one). Recall S is labelled



■ **Figure 2** Example of the query graph transformation.

by some subset of s vertices, $S = (u_{S,1}, \dots, u_{S,s})$, where we assume the sequence in which the $u_{S,i}$ are listed is consistent with some fixed topological order on all of G . Define sets

$$V_S := \left\{ v_{S,i}^{z_1, \dots, z_d} \mid i \in [s], z_1, \dots, z_d \in \{0, 1\}^s \right\} \quad (2)$$

and set $V' = \bigcup_{S \in V_T} V_S$. As depicted in Figure 2b, it will be helpful to continue to view V_S as a set, even though V_S is not a supervertex (i.e. G' itself will not be a separator tree). Intuitively, $v_{S,i}^{z_1, \dots, z_d}$ in V' represents node $u_{S,i}$ in V , but conditioned on “outcome strings” $z_1, \dots, z_d \in \{0, 1\}^s$ in the separators S_1, \dots, S_d . To formalize this relationship, we define a surjective function preimage: $V' \rightarrow V$ with preimage($v_{S,i}^{z_1, \dots, z_d}$) := $u_{S,i}$ for all S, i, z_1, \dots, z_d . Finally, since T has at most n supernodes, we have $|V'| \leq 2^{O(sD)}n$.

Next, define edges

$$E_S = \left\{ (v_{S,i}^{z_1, \dots, z_d}, v_{S_j,k}^{z_1, \dots, z_j}) \mid i \in [s], j \in [d-1], u_{S_j,k} \in \text{Desc}(u_{S,i}) \right\}, \quad (3)$$

where recall $u_{S_j,k} \in \text{Desc}(u_{S,i})$ is the set of all descendants of $u_{S,i}$ in the original graph G . In words, each E_S creates, for each copy of $u_{S,i}$, edges to all copies of descendants u_{S_j} which are on a strictly higher level in the separator tree (due to the $j \in [d-1]$ constraint). In the context of Figure 2a, this means we “shortcut” paths to descendants, but only via new edges pointing strictly “upwards” towards the root. Set $E' = \bigcup_{S \in V_T} E_S$. Observe that $|\text{Desc}(v)| \leq O(sD)$ for all $v \in V'$.

Assigning queries to G' . We have given a graph theoretic mapping $G \rightarrow G'$, but not yet specified how the queries made at nodes of G are mapped to queries made at nodes of G' . Let us do so now. Consider any $v_{S,i}^{z_1, \dots, z_d} \in V'$. Roughly, the goal is for the query at $v_{S,i}^{z_1, \dots, z_d}$

■ **Algorithm 2** Compute output of $u_{S,i}$, conditioned on results z_1, \dots, z_m in separators above.

```

1: function COMPUTEOUTPUT( $u_{S,i} \mid z_1, \dots, z_m$ )                                ▷ recall  $u_{S,i} \in S$ 
2:    $S_1, \dots, S_d \leftarrow$  path from the root to  $S$ 
3:   if  $m \geq d$  then                                                            ▷ base case of recursion; recursion has computed  $z_d$ 
4:     return  $z_{d,i}$                                                             ▷ recall  $z_d \in \{0, 1\}^s$ ;  $z_{d,i}$  encodes answer to  $u_{S,i}$ 
5:    $z_{m+1} \leftarrow 0^s$                                                         ▷ initialize answer bits to all zeroes to start
6:   for  $j = 1, \dots, s$  do                                                    ▷ in topological order, set answer bits at current level
7:      $z_{m+1,j} \leftarrow$  out-wire  $\left[ u_{S_{m+1,j}}^{z_1, \dots, z_{m+1}} \right]$     ▷ use query answers on incoming edges
8:   return COMPUTEOUTPUT( $u_{S,i} \mid z_1, \dots, z_{m+1}$ )

```

to simulate the query at preimage($v_{S,i}^{z_1, \dots, z_d}$) = $u_{S,i}$. However, $v_{S,i}^{z_1, \dots, z_d}$ is “conditioned” on bit strings z_1, \dots, z_d , so the simulation is not straightforward. To make this formal, we use Algorithm 2 as follows:

► **Rule 33.** For each edge $(u_{T,j}, u_{S,i})$ in G , the result of $\text{COMPUTEOUTPUT}(u_{T,j} \mid z_1, \dots, z_d)$ is used as the corresponding input to $v_{S,i}^{z_1, \dots, z_d} \in V'$.

Intuitively, we may view the conditioning string z_1, \dots, z_d as specifying a “parallel universe”, where if $(u_{T,j}, u_{S,i})$ was an edge in E , then this parent-child relationship is simulated *relative to this parallel universe* via COMPUTEOUTPUT (abbreviated as $\text{CO}()$ in the following).

► **Remark 34.** (1) By definition of a separator tree, all edges $(u_{T,j}, u_{S,i})$ of G must be in the same branch as S (i.e. either above or below S in the same branch, but not in a parallel branch of the tree). (2) This implies there are essentially two cases to consider: When $u_{T,j}$ is closer to the root than $u_{S,i}$, or vice versa. In the first case, Line 4 of Algorithm 2 immediately returns the hardcoded bit of z_1, \dots, z_d corresponding to $u_{T,j}$. In the second case, when $u_{S,i}$ calls Algorithm 2, Lines 6-8 will recursively compute outputs of nodes below $v_{S,i}^{z_1, \dots, z_d}$ in the same branch. For this, $v_{S,i}^{z_1, \dots, z_d}$ will need access to the out-wire functions (Definition 23) of certain nodes below it; this is afforded to $v_{S,i}^{z_1, \dots, z_d}$ via the edge set E_S (demonstrated via the “upward” black edges in Figure 2b).

We have now specified the local input/output behavior of any node $v \in V'$. Two problems remain: First, we require a designated output node in G' , which implicitly orchestrates the new logic in G' . Second, observe in Figure 2b that the original output node of G , y_2 , has been mapped to a new set of nodes labelled $y_2^{z_1, z_2, z_3}$, all of which are *disconnected* from the rest of G' . Thus, we require a mechanism to stitch together these components of G' . For that, we define G'' by adding a new output node, t , such that: (1) t has incoming edges from all nodes in V' , and (2) the output of G'' is computed by having t call $\text{CO}(v \mid \varepsilon)$ and return its answer, where ε denotes the empty string and v is the original output node of G . Both t and these new edges are depicted in Figure 2b via dashed lines. We refer to the full version for an extensive discussion of the intuition behind this construction.

4.2.2 Merging Nodes (G^*)

Next, we address the issue that G and G'' are not necessarily equivalent for promise problems, since copies of the same invalid query could have different outputs. For example, in Figure 2b, if $z_{1,1} = 1$, then u_1^{00} and $u_2^{z_1, 1^0}$ depend on different copies of v_2 , which could lead to inconsistencies if v_2 encodes an invalid query. In addressing this, we will also remove redundant copies of nodes (e.g. v_1 , which has in-degree 0 in Figure 2a, encodes the same query in Figure 2b, regardless of how z_1 and z_2 are set).

To proceed, we construct graph G^* by merging node copies which have the same hard-coded inputs. Consider any $u := u_{S,i} \in V$, where as in *Algorithm 2*, we let d denote the depth of S on the unique superpath $P_S := (S_1, \dots, S_d = S)$ from the root S_1 to S in the separator tree. Recalling that $\text{Anc}(u)$ denotes the ancestors of u in G , i.e. the set of queries u depends on, define $D_u := \text{Anc}(u) \cap \bigcup_{j=1}^d S_j$, in words, the ancestors of u in the superpath P_u . For any $v := v_{S,i}^{z_1, \dots, z_d} \in V''$, define $h_v : D_u \rightarrow \{0, 1\}$ with action $h_v(u_{S_j,k}) := z_{j,k}$, i.e. h_v selects out the hard-coded bit $z_{j,k}$ corresponding to any $u_{S_j,k} \in D_u$ (i.e. $z_{j,k}$ is the k th bit of z_j in the definition of v). Now, whenever two copies $v_1, v_2 \in V''$ of the same node (i.e. $\text{preimage}(v_1) = \text{preimage}(v_2)$) satisfy $h_{v_1} = h_{v_2}$ (i.e. h_{v_1} and h_{v_2} have the same truth table; note D_u is in the original graph G in definition $h_v : D_u \rightarrow \{0, 1\}$), we will merge them. Formally, the merge is accomplished by *Algorithm 3*, which simultaneously computes an admissible weighting function. Henceforth, denote $(G^*, f^*) := \text{MERGE}(G'')$.

■ **Algorithm 3** Merge nodes in G'' to compute G^* .

```

1: function MERGE( $G'' = (V'', E'')$ )
2:    $G_1 \leftarrow G'', V_1 \leftarrow V'', E_1 \leftarrow E'', f_1 \leftarrow \omega$  ▷ for weighting function  $\omega$  from Lemma 31
3:    $i \leftarrow 1$ 
4:   while  $\exists v_1, v_2 \in V_i$  such that  $\text{preimage}(v_1) = \text{preimage}(v_2)$  and  $h_{v_1} = h_{v_2}$  do
5:     Choose such  $v_1, v_2$  so that  $v := \text{preimage}(v_1)$  is furthest from root in  $T$ .
6:     Create copy  $v^*$  of  $v$  with  $h_{v^*} := h_{v_1} = h_{v_2}$ .
7:      $V_{i+1} \leftarrow V_i \setminus \{v_1, v_2\} \cup \{v^*\}$ .
8:     Replace out-wire[ $v_1$ ], out-wire[ $v_2$ ] in the logic of nodes in  $V_{i+1}$  with out-wire[ $v^*$ ].
9:      $E_{i+1} \leftarrow \{(r(x), r(y)) \mid (x, y) \in E_i\}$ , where  $r(x) := \begin{cases} v^*, & \text{if } x \in \{v_1, v_2\} \\ x, & \text{else} \end{cases}$ .
10:    Update  $f_{i+1} : V_{i+1} \rightarrow \mathbb{R}$  such that  $f_{i+1}(x) := \begin{cases} f_i(v_1) + f_i(v_2), & \text{if } x = v^* \\ f_i(x), & \text{else} \end{cases}$ .
11:  return  $(G_i, f_i)$ 

```

► **Remark 35.** When $u \in V$ has in-degree 0, then $D_u = \emptyset$. In this case, for any two copies $v_1, v_2 \in V''$ of u , it is vacuously true that $h_{v_1} = h_{v_2}$. Thus, all copies of u in V'' are merged by *Algorithm 3*. An example of this is depicted by v_1 in Figure 2a being mapped to $v_1^{*,**}$ in Figure 2c. Intuitively, this captures the fact that since v_1 is in-degree 0 in Figure 2a, the query at all copies $v_1^{z_1, z_2}$ of Figure 2b is identical, regardless of how z_1, z_2 are set.

► **Lemma 36.** *For any constant $c \geq 2$, the weighting function f^* produced by *Algorithm 3* is c -admissible for G^* , and satisfies $W_{f^*}(G^*) = W_\omega(G'')$ (for ω from Lemma 31).*

4.2.3 Correctness

We now prove correctness, in the process establishing the Compression Lemma (Lemma 32). For this, we first require the following lemma, which shows how to map any correct query string for G^* to a correct query string for G . Below, $\text{CO}()$ on G^* takes into account merged notes, i.e. it uses v^* instead of v after merging v_1 and v_2 in *Algorithm 3*.

► **Lemma 37.** *Let $x^* : V^* \rightarrow \{0, 1\}$ be a correct query string for G^* . Define $\text{CO}^*()$ to be $\text{CO}()$, except with each call to out-wire on Line 7 replaced by looking up the corresponding bit of x^* . Define string $x : V \rightarrow \{0, 1\}$ such that bit $x(v) := \text{CO}^*(v \mid \varepsilon)$. Then, x is a correct query string for G .*

Proof. Recall $|V| = n$, and that by Definition 24, a correct query string for C -DAG is defined as any string producible by Line 6 of Algorithm 1. Throughout, the bits of x are ordered according to the topological order (v_1, \dots, v_n) on V fixed by Algorithm 1. We prove the claim inductively for $t \in (1, \dots, n)$.

By the topological order, the base case $v_1 \in V$ has in-degree 0, i.e. takes no inputs. Thus, by Remark 35, there is only a single node in G^* corresponding to v_1 , which by construction computes the same query as v_1 . Hence, the corresponding bit of x^* trivially encodes the correct answer for v_1 in G and is then returned for v_1 once Line 4 executes, as desired.

For the inductive case, let $t \geq 2$ and assume x_1, \dots, x_{t-1} satisfy the induction hypothesis, i.e. they could be produced by the first $t-1$ iterations of EVALUATE. We now need to argue that a correct execution of EVALUATE could set $x_t = \text{CO}^*(v_t \mid \varepsilon)$. By design, $\text{CO}^*(v_t \mid \varepsilon)$ computes z_1, \dots, z_d and then returns⁷ $x^*(v_t^{z_1, \dots, z_d})$.

Recall now that D_{v_t} denotes the ancestors of v_t in G , which are also along the superpath from the root of the separator tree down to the supervertex S containing v_t . We claim that $z := z_1, \dots, z_d$ matches $x_1 \cdots x_{t-1}$ on D_{v_t} . (For clarity, the bits of $z \in \{0, 1\}^{sd}$ are ordered according to the recursion of $\text{CO}^*(\cdot)$ and may also contain bits corresponding to vertices not in D_{v_t} .) To see this, fix any $u \in D_{v_t} = \text{Anc}(v_t) \cap \bigcup_{j=1}^d S_j$, and let i be the supervertex index such that $u \in S_i$. For brevity, define notation $x(u)$ and $z(u)$ to mean the bit of x and z corresponding to u , respectively⁸. Now, recall $\text{CO}^*(v_t \mid \varepsilon)$ recursively traverses the path S_1, \dots, S_d , where $u \in S_i$ and $v_t \in S_d$ for $1 \leq i \leq d$. Thus, the operations performed by $\text{CO}^*(v_t \mid \varepsilon)$ and $\text{CO}^*(u \mid \varepsilon)$ are identical in the first i recursions. Once $m = i$ (i.e. conditioning strings z_1, \dots, z_i have been set), $z(u)$ is returned by $\text{CO}^*(u \mid \varepsilon)$ on Line 4, whereas $\text{CO}^*(v_t \mid \varepsilon)$ returns $z(v_t)$ if $i = d$ and continues recursively otherwise.

We thus conclude $\text{CO}^*(v_t \mid \varepsilon)$ returns $x^*(v_t^{z_1, \dots, z_d})$ with $z(u) = x(u)$ for all $u \in D_{v_t}$. Recall now by Rule 33 that, in order for $v_t^{z_1, \dots, z_d} \in V^*$ to simulate the query at $v_t \in V$, it computes the input on any wire u into v_t via $\text{CO}^*(u \mid z_1, \dots, z_d)$. Since the construction is based on a separator tree, this incoming wire/edge (u, v_t) lies along the same branch of the tree as v_t . Thus, we have only two cases to consider – $u \in \text{Anc}(v_t)$ is above or below v_t in said branch. (In Figure 2a, for example, if $v_t = w_2$, the ancestor u_2 is above w_2 in the tree, whereas ancestor y_1 is below w_2 .) So, if $u \in D_{v_t}$ (i.e. u is above D_{v_t}), then by the argument in the previous paragraph, $z(u) = x(u)$ is used as input to v_t . Moreover, since $u \in \text{Anc}(v_t)$, u comes before v_t in any topological order, and thus the induction hypothesis says $x(u)$ is correct. Otherwise, if $u \notin D_{v_t}$ (i.e. ancestor u is below D_{v_t}), then it again holds that $\text{CO}^*(v_t \mid \varepsilon)$ and $\text{CO}^*(u \mid \varepsilon)$ perform exactly the same operations in the first d recursions. Therefore, both executions compute the same values z_1, \dots, z_d . Subsequently, $\text{CO}^*(u \mid \varepsilon) = x(u)$ calls $\text{CO}^*(u \mid z_1, \dots, z_d)$ on Line 8 during the d th recursion and returns its value. In other words, $x(u) = \text{CO}^*(u \mid z_1, \dots, z_d)$. But by Rule 33, in order to simulate its input on the incoming wire corresponding to u , $v_t^{z_1, \dots, z_d}$ uses $\text{CO}^*(u \mid z_1, \dots, z_d) = x(u)$. Then, since $u \in \text{Anc}(v_t)$,

⁷ Technically, the algorithm actually returns $x^*(v_t^{z_1, \dots, z_{d-1}, z'_d})$, where $z'_d \in \{0, 1\}^s$ is an “intermediate string” defined as follows: If node v_t was the k th node in the topological order for supervertex S , then the first $k-1$ bits of z'_d have been assigned by Line 7 of $\text{CO}^*(\cdot)$, and the remaining $s-k+1$ bits of z'_d are still set to the dummy value of 0 from Line 5. However, this does not affect our analysis. In particular, in G^* we merged $v_t^{z_1, \dots, z'_d}$ and $v_t^{z_1, \dots, z_{d-1}, z''_d}$ for any such pair z'_d and z''_d of “intermediate strings”, since by definition of the topological order, bits k through s of any such z'_d cannot correspond to any vertices in D_{v_t} . Thus, these indices effectively disappear for all copies of v_t in G^* .

⁸ Since x is defined on G , $x(u)$ is clearly defined uniquely. On the other hand, z is defined on G^* , which contains potentially multiple copies of u ; thus, it is slightly more subtle that $z(u)$ is uniquely defined. Indeed, uniqueness holds since the recursive path followed by $\text{CO}^*(\cdot)$ through G^* visits *precisely one* copy of u ; *which* copy is visited depends on the prefix of z fixed in the recursion before u is encountered.

u comes before v_t in any topological order, and thus by the induction hypothesis, $x(u)$ is correct. We hence conclude all input wires to $v_t^{z_1, \dots, z_d}$ must be set correctly, and thus $x(v_t)$ is also correct. \blacktriangleleft

We can finally prove the main lemma of this section.

Proof of Lemma 32. G^* is constructed as in Section 4.2.1 and Section 4.2.2. We have $|V^*| \leq |V''| \leq 2^{O(sD)}n$ (recall $n = |V|$), as there are $\leq 2^{O(sD)}$ choices for conditioning strings $z_1 \cdots z_D$. Since we assume the separator tree is given as input, the time to construct G^* is clearly polynomial in $|V''|$, i.e. $2^{O(sD+\log n)}$. For weighting function ω from Lemma 31, we have $W_{f^*}(G^*) = W_\omega(G'') \leq (c+1)^{O(sD)}n$, where the equality is from Lemma 36, and the inequality because every node in G'' has at most $O(sD)$ descendants. Correctness follows from Lemma 37 and the fact that the output of G^* , by definition of node t , is $\text{CO}(v \mid \varepsilon)$. Finally, each verification circuit corresponding to a node in V^* has size $\text{poly}(|V^*|)$; the largest such verification circuit corresponds to node t , which takes in wires from *all* other vertices in G^* , and calls $\text{CO}(v \mid \varepsilon)$ (which takes time $\text{poly}(|V^*|)$). \blacktriangleleft

4.3 Solving C -DAG via oracle queries

We now show how to decide an N -node C -DAG-instance G with a c -admissible weighting function f using $O(\log W_f(G))$ oracle queries. (We intentionally use N to denote the size of G , to avoid confusion with the parameter n from Section 4.2.) Recall that at a high level, our aim is to convert the problem of deciding G into the problem of maximizing a carefully chosen real-valued function t . A binary search via the oracle C is then conducted to compute the optimal value to t , from which a correct query string from G can be extracted. This high-level strategy was also used by Gottlob [22]; what is different here is how we define t and how we implement the details of the binary search.

4.3.1 Step 1: Defining the total solution weight function t

Let $G = (V = \{v_1, \dots, v_N\}, E)$ be a C -DAG-instance with c -admissible weighting function f . Recall by Definition 23 that each circuit Q_i has a proof register \mathcal{Y}_i . Without loss of generality, we assume Q_i receives a proof $|\psi_i\rangle \in \mathcal{B}^{\otimes m}$ and has completeness α and soundness β . Then, define $t : \{0, 1\}^N \times (\mathcal{B}^{\otimes m})^{\times N} \rightarrow \mathbb{R}$ such that

$$t(x, \psi_1, \dots, \psi_N) := \sum_{i=1}^N f(v_i) \underbrace{(x_i \Pr[Q_i(z_i(x), \psi_i) = 1] + (1 - x_i)\gamma)}_{g(x_i, z_i(x), \psi_i)}, \quad (4)$$

where $\gamma := (\alpha + \beta)/2$, and $z_i(x)$ is defined similar to Line 5 of Algorithm 1, i.e. $z_i(x) \leftarrow \bigcirc_{v_j \in \text{parents}(v_i)} x_j$, for x the input string to t . Two comments are important here: First, defining $z(x)$ in this manner may break the logic of Algorithm 1 when a prover is dishonest, in that the relationship between x_i and z_i of Line 6 may not hold. Nevertheless, in Section 4.3.3, we prove that t is maximized only when a prover acts *honestly*. Second, we intentionally define t as taking in a *cross product* over spaces $\mathcal{B}^{\otimes m}$, as opposed to a tensor product. This simplifies the proofs of this section. Finally, define

$$T := \max_{x \in \{0,1\}^N, |\psi_1\rangle, \dots, |\psi_N\rangle \in \mathcal{B}^{\otimes m}} t(x, \psi_1, \dots, \psi_N). \quad (5)$$

4.3.2 Step 2: Approximating T

In order to apply binary search to approximate T (see proof of Theorem 28), we now show that the *decision* version of approximating T is in C . Namely, define promise problem $\Pi_\varepsilon = (\Pi_{\text{yes}}, \Pi_{\text{no}})$ such that $\Pi_{\text{yes}} = \{(t, s) \mid t : \{0, 1\}^N \times (\mathcal{B}^{\otimes m})^{\times N} \rightarrow \mathbb{R} \text{ and } T \geq s\}$ and $\Pi_{\text{no}} = \{(t, s) \mid t : \{0, 1\}^N \times (\mathcal{B}^{\otimes m})^{\times N} \rightarrow \mathbb{R} \text{ and } T \leq s - \varepsilon\}$, for T as in Equation (5), and $\varepsilon : \mathbb{Z} \rightarrow \mathbb{R}^{\geq 0}$ a fixed function of N (i.e. by ε we mean $\varepsilon(N)$).

► **Lemma 38.** *Let $C \in \mathcal{QV}^+$. Define $W := \sum_{i=1}^N f(v_i)$ for weighting function f from Equation (4), and assume $W \leq \text{poly}(N)$. Then, for any $\varepsilon \geq 1/\text{poly}(N)$, $\Pi_\varepsilon \in C$.*

Proof. In the case $C \in \{\text{NP}, \text{NEXP}\}$, Π_ε can easily be solved in C by just computing $t(x, \psi_1, \dots, \psi_N)$ directly (note that $t : \{0, 1\}^N \times (\{0, 1\}^{\otimes m})^{\times N} \rightarrow \mathbb{R}$ in this case).

For the remaining C , we begin by defining probabilities $p_i := f(v_i)/W$ and let

$$t'(x, \psi_1, \dots, \psi_N) := \frac{1}{W} t(x, \psi_1, \dots, \psi_N) = \sum_{i=1}^N p_i \cdot g(x_i, z_i, \psi_i), \quad (6)$$

whose maximum over all inputs we denote as T' . We prove the claim by constructing a C -verifier V such that

$$\max_{\text{proofs } |\psi\rangle} \Pr[V \text{ outputs } 1 \mid |\psi\rangle] = T'. \quad (7)$$

Thus, when $(t, s) \in \Pi_{\text{yes}}$ (resp., $(t, s) \in \Pi_{\text{no}}$), V accepts with probability at least s/W (resp., at most $(s - \varepsilon)/W$), where $\varepsilon/W \geq 1/\text{poly}(N)$ since $W \leq \text{poly}(N)$ by assumption.

V has proof space $\mathcal{X} \otimes \mathcal{Y}_1 \otimes \dots \otimes \mathcal{Y}_N$ with $\mathcal{X} = \mathcal{B}^{\otimes N}$ and $\mathcal{Y}_i = \mathcal{B}^{\otimes m}$. A subtle point here is that function t takes as part of its input a sequence $(|\psi_1\rangle, \dots, |\psi_N\rangle)$, whereas in Equation (7), V takes in a joint (potentially entangled) proof $|\psi\rangle$ across proof registers $\mathcal{Y}_1 \otimes \dots \otimes \mathcal{Y}_N$. However, due to the construction of V below, we shall see that without loss of generality, Equation (7) is attained for tensor product states $|\psi\rangle = |\psi_1\rangle \otimes \dots \otimes |\psi_N\rangle$, which is equivalent to sequence $(|\psi_1\rangle, \dots, |\psi_N\rangle)$, as desired.

Given proof $|\psi\rangle \in \mathcal{X} \otimes \mathcal{Y}_1 \otimes \dots \otimes \mathcal{Y}_N$, V acts as follows:

- 1: Measure \mathcal{X} in standard basis to obtain string x .
- 2: Select random i according to distribution p_i .
- 3: **if** $x_i = 1$ **then**
- 4: Run Q_i with input $z_i(x)$ and proof register \mathcal{Y}_i .
- 5: **else**
- 6: Output 1 with probability γ .

Since (the POVM corresponding to) V is block diagonal with respect to \mathcal{X} , $\Pr[V \text{ outputs } 1 \mid |\psi\rangle]$ is maximized by some $|\psi\rangle = |x\rangle_{\mathcal{X}} |\psi'\rangle_{\mathcal{Y}_1, \dots, \mathcal{Y}_N}$. Then, since we only measure a single local verifier Q_i (at random, Step 4), we have

$$\Pr[V \text{ outputs } 1 \mid |\psi\rangle] = t'(x, \sigma_1, \dots, \sigma_N) \quad \text{where} \quad \sigma_i := \text{Tr}_{\bigotimes_{j \neq i} \mathcal{Y}_j} (|\psi'\rangle \langle \psi'|). \quad (8)$$

But for any fixed x , this is maximized by choosing pure states $\sigma_i = |\psi_i\rangle \langle \psi_i|$. Thus, t' is optimized by a tensor product $|\psi\rangle = |\psi_1\rangle \otimes \dots \otimes |\psi_N\rangle$, and so Equation (7) holds. This completes the proof.⁹ ◀

⁹ See the full version for additional details on StoqMA and QMA(2).

4.3.3 Step 3: Correct Query String

We next show that only *correct* query strings x can attain T (even *approximately*).

► **Lemma 39.** *Define $\eta := (\alpha - \beta)/2$, and let f be η^{-1} -admissible. If $t(x, \psi_1, \dots, \psi_N) > T - \eta$, then x is a correct query string.*

Proof sketch. Assume there exists a $v_i \in V$ such that x_i is incorrect. We show that there exist $x', |\psi'_1\rangle, \dots, |\psi'_N\rangle \in \mathcal{B}^{\otimes N}$ such that $t(x', \psi'_1, \dots, \psi'_N) \geq t(x, \psi_1, \dots, \psi_N) + \eta$, obtaining a contradiction. Define $x', |\psi'_1\rangle, \dots, |\psi'_N\rangle$ such that $x'_i = \bar{x}_i$ (i.e. the complement of x_i), $x'_j = x_j$ and $|\psi'_j\rangle = |\psi_j\rangle$ for $j \neq i$, and $|\psi'_i\rangle$ maximizing $\Pr[Q_i(z_i(x'), \psi'_i) = 1]$. It is straightforward to show that $g(x'_i, z_i(x), \psi'_i) - g(x_i, z_i(x), \psi_i) \geq \eta$. Then,

$$t(x', \psi'_1, \dots, \psi'_N) - t(x, \psi_1, \dots, \psi_N) \geq \eta \left(f(v_i) - \eta^{-1} \sum_{(v_i, v_j) \in E} f(v_j) \right) \geq \eta, \quad (9)$$

since $g(\cdot) \in [0, 1]$, flipping x_i only affects children of v_i , and f is η^{-1} -admissible. ◀

4.3.4 Step 4: Completing the Proof

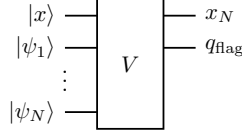
We now combine everything to show the main technical result of Section 4, Theorem 28.

Proof of Theorem 28. First, apply the Compression Lemma (Lemma 32) to transform G into an equivalent G^* with $|V^*| \leq 2^{O(sD)}n$ and $W_{f^*}(G^*) \leq (c + 1)^{O(sD)}n$. This takes $2^{O(sD + \log n)}$ time. Second, define the total solution weight function t as in Equation (4), whose maximum value we denoted T (Equation (5)). By Lemma 39, we know that any query string x satisfying $t(x, \psi_1, \dots, \psi_n) > T - \eta$ (for $\eta = (\alpha - \beta)/2$, α and β the completeness/soundness parameters for each C -verifier Q_i in the C -DAG, and for $f = f^*$ a η^{-1} -admissible weighting function) is a correct query string. So, assume without loss of generality (since $C \in \{\text{NP}, \text{MA}, \text{QCMA}, \text{QMA}, \text{QMA}(2)\}$, where for QMA(2) we use [25]) that $\alpha = 2/3$ and $\beta = 1/3$, so that $\eta^{-1} = 6$. By Lemma 36, f^* is c -admissible for any $c \geq 2$, and hence η^{-1} -admissible. Third, use Lemma 38 in conjunction with binary search to approximate T for G^* . Here, we must be slightly careful. Set $N = |V^*| \leq 2^{O(sD)}n$. Since the precision parameter $\eta \in \Theta(1)$, it suffices to use $\log(W_{f^*}(G^*)) \in O(\log|V^*|) \in O(sD + \log n)$ C -queries to resolve T within additive error η . Let \tilde{T} denote this estimate of T . Fourth, make a final C -query via Lemma 38 to decide whether there exists a correct query string x and proofs $|\psi_1\rangle, \dots, |\psi_N\rangle$, such that $t(x, \psi_1, \dots, \psi_N) \geq \tilde{T}$ and for which $x_N = 1$, and return its answer. (Recall that x_N , by definition, encodes the output of the C -DAG.) ◀

5 Hardness for APX-SIM via a unified framework

We now show how the construction of Section 4 can be embedded directly into the flag-qubit Hamiltonian construction of [46], thus directly yielding hardness results for the APX-SIM problem (Definition 17).

Definitions. The following definitions were introduced in [46] to allow one to abstractly speak about large classes of circuit-to-Hamiltonian mappings. This allows the Lifting Lemma of [46], as well as its generalized version shown in Section 5.1 (Lemma 42), to be used in a black-box fashion (i.e. agnostic to the particular choice circuit-to-Hamiltonian construction used). As a result, both Lifting Lemmas automatically preserve desirable properties of the actual circuit-to-Hamiltonian mappings employed, such as being 1D or translation invariant.



■ **Figure 3** Depiction of the circuit V constructed in Lemma 38, with two minor modifications: (1) The second wire denotes the output wire of V , and is relabeled q_{flag} here. (2) We assume without loss of generality that V outputs the N th bit of $x \in \{0, 1\}^N$ on the first wire above, labelled x_N .

► **Definition 40** (Conformity [46]). *Let H be a Hamiltonian with some well-defined structure S (such as k -local interactions, all constraints drawn from a fixed finite family, with a fixed geometry such as 1D, translational invariance, etc). We say a Hermitian operator P conforms to H if $H + P$ also has structure S .*

► **Definition 41** (Local Circuit-to-Hamiltonian Mapping [46]). *Let $\mathcal{X} = (\mathbb{C}^2)^{\otimes p}$ and $\mathcal{Y} = (\mathbb{C}^2)^{\otimes q}$. A map $H_w : \mathcal{U}(\mathcal{X}) \rightarrow \text{Herm}(\mathcal{Y})$ is a local circuit-to-Hamiltonian mapping if, for any $L > 0$ and any sequence of 2-qubit unitary gates $U = U_L U_{L-1} \cdots U_1$, the following hold:*

1. (Overall structure) $H_w(U) \succeq 0$ has a non-trivial null space, i.e. $\text{Null}(H_w(U)) \neq 0$. This null space is spanned by (some appropriate notion of) “correctly initialized computation history states”, i.e. with ancillae qubits set “correctly” and gates in U “applied” sequentially.
2. (Local penalization and measurement) Let q_1 and q_2 be the first two output wires of U (each a single qubit), respectively. Let $S_{\text{pre}} \subseteq \mathcal{X}$ and $S_{\text{post}} \subseteq \mathcal{Y}$ denote the sets of input states to U satisfying the structure enforced by $H_w(U)$ (e.g. ancillae initialized to zeroes), and null states of $H_w(U)$, respectively. Then, there exist projectors M_1 and P_L , projector M_2 conforming to $H_w(U)$, and a bijection $f : S_{\text{pre}} \rightarrow S_{\text{post}}$, such that for all $i \in \{1, 2\}$ and $|\phi\rangle \in S_{\text{pre}}$, the state $|\psi\rangle = f(|\phi\rangle)$ satisfies

$$\text{Tr}(|0\rangle\langle 0|_i (U_L U_{L-1} \cdots U_1) |\phi\rangle\langle \phi| (U_L U_{L-1} \cdots U_1)^\dagger) = \text{Tr}(|\psi_L\rangle\langle \psi_L| M_i), \quad (10)$$

where $|\psi_L\rangle = P_L |\psi\rangle / \|P_L |\psi\rangle\|_2$ is $|\psi\rangle$ postselected on measurement outcome P_L (we require $P_L |\psi\rangle \neq 0$). Moreover, there exists a function $g : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{R}$ such that

$$\|P_L |\psi\rangle\|_2^2 = g(p, L) \text{ for all } |\psi\rangle \in \text{Null}(H_w(U)), \quad \text{and} \quad M_i = P_L M_i P_L. \quad (11)$$

The map H_w , and all operators/functions above (M_1, M_2, P_L, f, g) are computable given U .

5.1 The Generalized Lifting Lemma

► **Lemma 42** (Generalized Lifting Lemma for APX-SIM). *Fix $C \in \mathcal{QV}^+$. As input, we are given a C-DAG instance G^* on N nodes, and c -admissible weighting function f^* . Let V , as depicted in Figure 3, be the verification circuit constructed in Lemma 38, given (G^*, f^*) . Define shorthand Δ for $\Delta(H_w(V))$. Fix a local circuit-to-Hamiltonian mapping H_w , and assume the notation in Definition 41. Fix any function $\alpha : \mathbb{N} \rightarrow \mathbb{N}$ such that $\alpha > \max\left(\frac{4\|M_2\|}{\Delta}, \frac{\Delta}{3\|M_2\|^2}, 1\right)$. Then, the Hamiltonian $H := \alpha H_w(V) + M_2$ satisfies:*

- If G is a YES instance, then for all $|\psi\rangle$ with $\langle \psi | H | \psi \rangle \leq \lambda_{\min}(H) + \frac{1}{\alpha^2}$,
$$\langle \psi | M_1 | \psi \rangle \leq \frac{1}{\alpha} \left[\frac{W}{\eta} \left(\frac{1}{\alpha} + \frac{12\|M_2\|^2}{\Delta} \right) + \frac{12\|M_2\|^2}{\Delta} \right].$$
- If G is a NO instance, then for all $|\psi\rangle$ with $\langle \psi | H | \psi \rangle \leq \lambda_{\min}(H) + \frac{1}{\alpha^2}$,
$$\langle \psi | M_1 | \psi \rangle \geq g(p, L) - \frac{1}{\alpha} \left[\frac{W}{\eta} \left(\frac{1}{\alpha} + \frac{12\|M_2\|^2}{\Delta} \right) - \frac{12\|M_2\|^2}{\Delta} \right],$$

for W and η defined in Lemma 38 and Lemma 39, respectively, and $g(p, L)$ defined in Definition 41.

Proof. The claim follows immediately from Lemmas 43–45 with $\delta := 1/\alpha^2$. \blacktriangleleft

We next give the three lemmas required for the proof of Lemma 42, all of which assume the notation for the latter.

► **Lemma 43** ([46]). *Fix any function $\alpha : \mathbb{N} \rightarrow \mathbb{N}$ such that $\alpha > \max\left(\frac{4\|M_2\|}{\Delta}, \frac{\Delta}{3\|M_2\|^2}, 1\right)$, and any $\delta \leq 1/\alpha^2$. Then, for any $|\psi\rangle$ such that $\langle\psi|H|\psi\rangle \leq \lambda_{\min}(H) + \delta$, there exists a uniform history state $|\phi\rangle \in \text{Null}(H_w(V))$ such that $\| |\psi\rangle\langle\psi| - |\phi\rangle\langle\phi| \|_{\text{tr}} \leq \frac{12\|M_2\|}{\alpha\Delta}$ and where $|\phi\rangle$ has energy $\langle\phi|H|\phi\rangle \leq \lambda_{\min}(H) + \delta + \frac{12\|M_2\|^2}{\alpha\Delta}$.*

For the second lemma, Lemma 44, recall V has proof space $\mathcal{X} \otimes \mathcal{Y}_1 \otimes \cdots \otimes \mathcal{Y}_N$ with $\mathcal{X} = \mathcal{B}^{\otimes N}$ and $\mathcal{Y}_i = \mathcal{B}^{\otimes m}$. Henceforth, we denote an arbitrary (potentially entangled) proof in this space as $|w_{\mathcal{X}\mathcal{Y}}\rangle$. We remark Lemma 44 is our version of Lemma 23 of [46]; however, our proof is significantly simplified, despite our lifting lemma allowing arbitrary C-DAGs, due to the specific design of our verifier V from Lemma 38.

► **Lemma 44.** *Suppose history state $|\phi\rangle \in \text{Null}(H_w(V))$ has preimage $|\psi_{\text{in}}\rangle = f^{-1}(|\phi\rangle)$ (for bijection f from Definition 41), where $|\psi_{\text{in}}\rangle$ has proof $|w_{\mathcal{X}\mathcal{Y}}\rangle$ with total amplitude p_{bad} on incorrect query strings in \mathcal{X} . Then, $\langle\phi|H|\phi\rangle > \lambda_{\min}(H) + g(p, L) \frac{p_{\text{bad}} \cdot \eta}{W}$.*

Proof. Let $|\psi_{\text{out}}\rangle = V|\psi_{\text{in}}\rangle$. Letting X_+ and X_- denote the sets of correct and incorrect query strings, respectively, we may write

$$|w_{\mathcal{X}\mathcal{Y}}\rangle = \sum_{x \in X_-} \alpha_x |x\rangle_{\mathcal{X}} |\psi_x\rangle_{\mathcal{Y}} + \sum_{x \in X_+} \alpha_x |x\rangle_{\mathcal{X}} |\psi_x\rangle_{\mathcal{Y}}, \quad (12)$$

for $\sum_{x \in X_+ \cup X_-} |\alpha_x|^2 = 1$, arbitrary unit vectors $\{|\psi_x\rangle\}_x$, and $p_{\text{bad}} := \sum_{x \in X_-} |\alpha_x|^2$. Recall from Definition 41 that M_2 simulates the projector $|0\rangle\langle 0|_{q_{\text{flag}}}$ via

$$\text{Tr}(|0\rangle\langle 0|_2 (U_L U_{T-1} \cdots U_1) |\psi_{\text{in}}\rangle\langle\psi_{\text{in}}| (U_L U_{T-1} \cdots U_1)^\dagger) = \text{Tr}(|\phi_L\rangle\langle\phi_L| M_2), \quad (13)$$

(since we assumed in Figure 3 that the second output qubit of V is the flag qubit), where $|\phi_L\rangle$ is the history state $|\phi\rangle$ projected down onto time step T . We thus have

$$\langle\phi|H|\phi\rangle = \langle\phi|M_2|\phi\rangle = g(p, L) \langle\phi_L|M_2|\phi_L\rangle = g(p, L) \text{Tr}(|\psi_{\text{out}}\rangle\langle\psi_{\text{out}}| \cdot |0\rangle\langle 0|_{q_{\text{flag}}}) \quad (14)$$

$$= g(p, L) \text{Pr}[V \text{ rejects} \mid |w_{\mathcal{X}\mathcal{Y}}\rangle], \quad (15)$$

where the second statement follows from Equation (11), and the third from Equation (13). By Equation (5) and Equation (6), there exists a proof $|w'_{\mathcal{X}\mathcal{Y}}\rangle = |x\rangle|\psi_1\rangle \cdots |\psi_N\rangle$ accepted by V with probability precisely T/W . Let $|\psi'_{\text{in}}\rangle$ be an input state containing this optimal proof $|w'_{\mathcal{X}\mathcal{Y}}\rangle$. Lemma 39 now yields¹⁰

$$\langle\phi|H|\phi\rangle > g(p, L) \left(\text{Pr}[V \text{ rejects} \mid |w'_{\mathcal{X}\mathcal{Y}}\rangle] + \sum_{x \in X_-} |\alpha_x|^2 \frac{\eta}{W} \right) \quad (16)$$

$$= \langle\phi'|M_2|\phi'\rangle + g(p, L) \frac{p_{\text{bad}} \cdot \eta}{W} \geq \lambda_{\min}(H) + g(p, L) \frac{p_{\text{bad}} \cdot \eta}{W}, \quad (17)$$

where the first inequality (16) uses the fact that

$$\text{Pr}[V \text{ accepts} \mid |w_{\mathcal{X}\mathcal{Y}}\rangle] \leq p_{\text{good}} \cdot \frac{T}{W} + p_{\text{bad}} \left(\frac{T}{W} - \frac{\eta}{W} \right) = \frac{T}{W} - \frac{p_{\text{bad}} \cdot \eta}{W}. \quad (18)$$

The second statement uses Equation (11), with $|\phi'\rangle := f(|\psi'_{\text{in}}\rangle)$, and the last statement (17) uses $|\phi'\rangle \in \text{Null}(H_w(V))$ by the definition of f in Definition 41. \blacktriangleleft

¹⁰We are implicitly using the fact that, as observed in the proof of Lemma 38, for any fixed query string x , the acceptance probability of V is maximized by choosing a product state proof $|\psi_1\rangle \cdots |\psi_N\rangle$ on \mathcal{Y} .

Finally, the third lemma, Lemma 45, is our analog of Lemma 25 of [46]. We follow the same high-level approach as the latter, but again, our proof here is simplified. This is because Lemma 39 can be directly leveraged to obtain that any history state close enough to the ground space of H must simply output the correct answer to the input C -DAG on wire x_N in Figure 3. (In contrast, [46] needed the Commutative Quantum Union Bound to argue that all proofs are simultaneously correct.)

- **Lemma 45.** *Consider any $|\psi\rangle$ satisfying $\langle\psi|H|\psi\rangle \leq \lambda_{\min}(H) + \delta$ for $\delta \leq 1/\alpha^2$.*
- *If G^* is a YES instance, then $\langle\psi|M_1|\psi\rangle \leq \frac{W}{\eta} \left(\delta + \frac{12\|M_2\|^2}{\alpha\Delta} \right) + \frac{12\|M_2\|^2}{\alpha\Delta}$.*
 - *If G^* is a NO instance, then $\langle\psi|M_1|\psi\rangle \geq g(p, L) - \frac{W}{\eta} \left(\delta + \frac{12\|M_2\|^2}{\alpha\Delta} \right) - \frac{12\|M_2\|^2}{\alpha\Delta}$.*

Proof sketch. We use Lemma 43 to map $|\psi\rangle$ to a history state $|\phi\rangle \in \text{Null}(H_w(V))$ whose preimage $|\phi_{\text{in}}\rangle = f^{-1}(|\phi\rangle)$ contains proof $|w_{\mathcal{X}\mathcal{Y}}\rangle$ with amplitude at most p_{bad} on incorrect query strings in \mathcal{X} . By Equations (10) and (11), we have $\langle\phi|M_1|\phi\rangle = g(p, L)\langle\phi_L|M_1|\phi_L\rangle = g(p, L) \text{Tr}(|0\rangle\langle 0|_1 V|\phi_{\text{in}}\rangle\langle\phi_{\text{in}}|V^\dagger)$ (V in Figure 3 outputs x_N on the first wire). Combining this with the bounds from Lemmas 43 and 44 via Hölder's inequality yields the claim. ◀

5.2 Applying the Lifting Lemma

We now give two examples of how to use Lemma 42 to obtain hardness results for APX-SIM, for the cases of $C = \text{QMA}$ and $C = \text{StoqMA}$. The theorem below sets $N := \min(2^{O(s(n)\log n)}, 2^{O(d(n)\log n)})$ – the two values in $\min(\cdot, \cdot)$ correspond to the use of the bounded separator framework (Theorem 1) or bounded depth framework (Theorem 5), respectively, in conjunction with Lemma 42.

- **Theorem 46** (Hardness of APX-SIM for $C = \text{QMA}$ via Lemma 42). *Fix $C = \text{QMA}$, and let G be any C -DAG instance on n nodes with separator number and depth scaling as $s(n)$ and $d(n)$, in the sense of $C\text{-DAG}_s$ and $C\text{-DAG}_d$, respectively. Set $N := \min(2^{O(s(n)\log n)}, 2^{O(d(n)\log n)})$. Then, there exists a $\text{poly}(N)$ -time many-one reduction from G to an instance (H, a, b, δ) of APX-SIM, which satisfies: (1) H has size $\text{poly}(N)$ (i.e. acts on $\text{poly}(N)$ qubits/qudits, and has $\text{poly}(N)$ local terms), (2) H is either 5-local acting on qubits or 2-local on a 1D chain of 8-dimensional qudits (depending on which circuit-to-Hamiltonian mapping is employed), (3) $b - a \geq 1/\text{poly}(N)$ and $\delta \geq 1/\text{poly}(N)$.*

Proof sketch. Combine Lemmas 21, 32, and 42 with Kitaev's 5-local [32] or Hallgren, Nagaj, and Narayanaswami's 1D construction [24]. ◀

As noted in Section 1.1, combining Theorem 1 with Theorem 46, we have that $C\text{-DAG}_1$ can directly be embedded into an instance of APX-SIM.

- **Theorem 47** (Hardness of APX-SIM for $C = \text{StoqMA}$ via Lemma 42). *Fix $C = \text{StoqMA}$ and any efficiently computable $s : \mathbb{N} \rightarrow \mathbb{N}$, and define $N := \min(2^{O(s(n)\log^2 n)}, 2^{O(d(n)\log n)})$. Then, there exists a $\text{poly}(N)$ -time many-one reduction from any instance of $C\text{-DAG}$ to an instance (H, a, b, δ) of APX-SIM for stoquastic H , which satisfies: (1) H has size $\text{poly}(N)$ (i.e. acts on $\text{poly}(N)$ qubits, and has $\text{poly}(N)$ local terms), (2) H is 2-local, (3) $b - a \geq 1/\text{poly}(N)$ and $\delta \geq 1/\text{poly}(N)$.*

Thus, in the $N = 2^{O(d(n)\log n)}$ case (i.e. bounded depth framework), we recover that APX-SIM on stoquastic Hamiltonians is $\text{P}^{\text{StoqMA}[\log]}$ -hard [18]. For clarity, this follows because $\text{P}^{\text{StoqMA}[\log]} = \text{P}^{\parallel\text{StoqMA}}$ [18], and $\text{P}^{\parallel\text{StoqMA}}$ corresponds to a depth-1 StoqMA-DAG.

6 No-go statement for “weak compression” of polynomials

We observe that the weighting function approach applied to NP queries (introduced in [22] and used here as well) can be turned upside-down to obtain a no-go statement about a purely mathematical question: *Can arbitrary multi-linear polynomials be “weakly compressed”?*

► **Definition 48** (Weak compression of polynomials). *Let $f : [0, 1]^m \rightarrow \mathbb{R}^+$ be a multi-variate polynomial with rational coefficients, specified via an arithmetic circuit of size M . Assume there exists $x^* \in [0, 1]^m$ maximizing f such that $f(x)$ can be specified exactly¹¹ via B bits, for some (finite) B . We say f is weakly compressible to B' bits if there exists an efficiently computable mapping taking f to another function $g : [0, 1]^{m'} \rightarrow \mathbb{R}^+$ such that:*

1. *For any $y \in [0, 1]^{m'}$, $g(y)$ is computable in $\text{poly}(m)$ time.*
2. *(Optimality preserved) For any optimal y^* maximizing $g(y^*)$ over $[0, 1]^{m'}$, there exists a $\text{poly}(m)$ -time map taking y^* to an optimal $x^* \in [0, 1]^m$ maximizing $f(x^*)$.*
3. *(Compression) There exists an optimal y^* requiring at most B' bits to specify exactly.*

Next, we state the main technical lemma needed to show Lemma 9. Its proof along with those of Lemma 9 and Corollaries 10 and 11 are in the full version.

► **Lemma 49.** *Let t be as in Equation (4), specified using n bits of precision (used to describe weights w_i and verifiers V_i). There exists a poly-time Turing machine which, given t , produces an arithmetic circuit encoding multi-linear polynomial $p_{\text{out}} : [0, 1]^{\text{poly}(n)} \rightarrow \mathbb{R}^+$ with rational coefficients such that*

$$\max_{x, y_1, \dots, y_m \in \{0, 1\}^{\text{poly}(m)}} t(x, y_1, \dots, y_m) = \max_{s \in [0, 1]^{\text{poly}(m)}} p_{\text{out}}(s). \quad (19)$$

(Both f and p_{out} have range $[0, \sum_i |w_i|]$ over their respective domains.) Moreover, given an optimal s^ maximizing p_{out} , one can efficiently compute a correct NP query string for the NP-DAG underlying t .*

References

- 1 Scott Aaronson. BQP and the polynomial hierarchy. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing, STOC '10*, pages 141–150, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1806689.1806711.
- 2 Dorit Aharonov, Daniel Gottesman, Sandy Irani, and Julia Kempe. The power of quantum systems on a line. *Communications in Mathematical Physics*, 287(1):41–65, April 2009. doi:10.1007/s00220-008-0710-3.
- 3 Dorit Aharonov, Alex B. Grilo, and Yupan Liu. StoqMA vs. MA: the power of error reduction, 2021. arXiv:2010.02835.
- 4 Dorit Aharonov and Sandy Irani. Hamiltonian complexity in the thermodynamic limit, 2021. arXiv:2107.06201.
- 5 A. Ambainis. On physical problems that are slightly more difficult than QMA. In *2014 IEEE 29th Conference on Computational Complexity (CCC)*, pages 32–43, 2014.
- 6 Richard Beigel, L.A. Hemachandra, and G. Wechsung. On the power of probabilistic polynomial time: $\text{P}^{\text{NP}[\log]} \subseteq \text{PP}$. In *[1989] Proceedings. Structure in Complexity Theory Fourth Annual Conference*, pages 225–227, June 1989. doi:10.1109/SCT.1989.41828.
- 7 Hans L. Bodlaender, John R. Gilbert, Hjálmtýr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *J. Algorithms*, 18(2):238–255, March 1995. doi:10.1006/jagm.1995.1009.

¹¹ For clarity, we are assuming a naive binary expansion of $f(x^*)$.

- 8 Adam D. Bookatz. QMA-complete problems. *Quantum Information & Computation*, 14(5&6):361–383, April 2014.
- 9 Sergey Bravyi, Arvid J. Bessen, and Barbara M. Terhal. Merlin-Arthur Games and Stoquastic Complexity, December 2006. [arXiv:quant-ph/0611021](https://arxiv.org/abs/quant-ph/0611021).
- 10 Samuel R. Buss and Louise Hay. On truth-table reducibility to SAT. *Information and Computation*, 91(1):86–102, 1991. doi:10.1016/0890-5401(91)90075-D.
- 11 Jorge Castro and Carlos Seara. Characterizations of some complexity classes between ω_2^p and δ_2^p . In Alain Finkel and Matthias Jantzen, editors, *STACS 92*, Lecture Notes in Computer Science, pages 303–317, Berlin, Heidelberg, 1992. Springer. doi:10.1007/3-540-55210-3_192.
- 12 André Chailloux and Or Sattath. The Complexity of the Separable Hamiltonian Problem. In *2012 IEEE 27th Conference on Computational Complexity*, pages 32–41, June 2012. ISSN: 1093-0159. doi:10.1109/CCC.2012.42.
- 13 Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, Shaker Heights, Ohio, USA, May 1971. Association for Computing Machinery. doi:10.1145/800157.805047.
- 14 Toby Cubitt and Ashley Montanaro. Complexity classification of local hamiltonian problems. *SIAM Journal on Computing*, 45(2):268–316, 2016. doi:10.1137/140998287.
- 15 Bill Fefferman and Cedric Lin. Quantum Merlin Arthur with Exponentially Small Gap, January 2016. [arXiv:1601.01975](https://arxiv.org/abs/1601.01975).
- 16 Michael R. Garey and David. S. Johnson. *COMPUTERS and INTRACTABILITY: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.
- 17 Sevag Gharibian, Yichen Huang, Zeph Landau, and Seung Woo Shin. Quantum Hamiltonian Complexity. *Foundations and Trends® in Theoretical Computer Science*, 10(3):159–282, October 2015. doi:10.1561/04000000066.
- 18 Sevag Gharibian, Stephen Piddock, and Justin Yirka. Oracle Complexity Classes and Local Measurements on Physical Hamiltonians. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*, volume 154 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:37, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2020.20.
- 19 Sevag Gharibian and Dorian Rudolph. On polynomially many queries to NP or QMA oracles, 2021. [arXiv:2111.02296](https://arxiv.org/abs/2111.02296).
- 20 Sevag Gharibian and Justin Yirka. The complexity of simulating local measurements on quantum systems. *Quantum*, 3:189, September 2019. doi:10.22331/q-2019-09-30-189.
- 21 Daniel Gottesman and Sandy Irani. The quantum and classical complexity of translationally invariant tiling and hamiltonian problems. *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 95–104, 2009.
- 22 Georg Gottlob. NP trees and Carnap’s modal logic. *Journal of the ACM*, 42(2):421–457, March 1995. doi:10.1145/201019.201031.
- 23 Hermann Gruber. On balanced separators, treewidth, and cycle rank. *Journal of Combinatorics*, 3(4):669–681, 2012. doi:10.4310/JOC.2012.v3.n4.a5.
- 24 Sean Hallgren, Daniel Nagaj, and Sandeep Narayanaswami. The local hamiltonian problem on a line with eight states is QMA-complete. *Quantum Info. Comput.*, 13(9–10):721–750, September 2013.
- 25 Aram W. Harrow and Ashley Montanaro. Testing product states, quantum Merlin-Arthur games and tensor optimisation. *Journal of the ACM*, 60(1):1–43, February 2013. doi:10.1145/2432622.2432625.
- 26 Juris Hartmanis. Sparse complete sets for NP and the optimal collapse of the polynomial hierarchy. In *Current Trends in Theoretical Computer Science*, pages 403–411. WORLD SCIENTIFIC, June 1993. doi:10.1142/9789812794499_0029.
- 27 Lane A. Hemachandra. The strong exponential hierarchy collapses. *Journal of Computer and System Sciences*, 39(3):299–322, December 1989. doi:10.1016/0022-0000(89)90025-1.

- 28 Edith Hemaspaandra, Lane A. Hemaspaandra, and Jörg Rothe. Exact analysis of Dodgson elections: Lewis Carroll's 1876 voting system is complete for parallel access to NP. In Pierpaolo Degano, Roberto Gorrieri, and Alberto Marchetti-Spaccamela, editors, *Automata, Languages and Programming*, Lecture Notes in Computer Science, pages 214–224, Berlin, Heidelberg, 1997. Springer. doi:10.1007/3-540-63165-8_179.
- 29 Richard M. Karp. Reducibility among Combinatorial Problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, Boston, MA, 1972. doi:10.1007/978-1-4684-2001-2_9.
- 30 Julia Kempe, Alexei Kitaev, and Oded Regev. The complexity of the local hamiltonian problem. *SIAM Journal on Computing*, 35(5):1070–1097, January 2006. doi:10.1137/s0097539704445226.
- 31 Julia Kempe and Oded Regev. 3-local Hamiltonian is QMA-complete. *Quantum Information & Computation*, 3(3):258–264, May 2003.
- 32 A. Yu. Kitaev, A. H. Shen, and M. N. Vyalyi. *Classical and Quantum Computation*. American Mathematical Society, USA, 2002.
- 33 Mark W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490–509, 1988. doi:10.1016/0022-0000(88)90039-6.
- 34 Mark W. Krentel. Generalizations of Opt P to the polynomial hierarchy. *Theoretical Computer Science*, 97(2):183–198, April 1992. doi:10.1016/0304-3975(92)90073-0.
- 35 L. A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.
- 36 Tobias J. Osborne. Hamiltonian complexity. *Reports on Progress in Physics*, 75(2):022001, January 2012. doi:10.1088/0034-4885/75/2/022001.
- 37 Christos H. Papadimitriou. On the complexity of unique solutions. In *23rd Annual Symposium on Foundations of Computer Science (SFCS 1982)*, pages 14–20, 1982. doi:10.1109/SFCS.1982.28.
- 38 Christos H. Papadimitriou and Stathis K. Zachos. Two remarks on the power of counting. In Armin B. Cremers and Hans-Peter Kriegel, editors, *Theoretical Computer Science*, Lecture Notes in Computer Science, pages 269–275, Berlin, Heidelberg, 1982. Springer. doi:10.1007/BFb0036487.
- 39 Ran Raz and Avishay Tal. Oracle separation of bqp and ph. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 13–23, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316315.
- 40 Neil Robertson and P.D Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.
- 41 Philippe Schnoebelen. Oracle circuits for branching-time model checking. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *Automata, Languages and Programming: 30th International Colloquium, ICALP 2003 Eindhoven, The Netherlands, June 30 – July 4, 2003 Proceedings*, pages 790–801, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. doi:10.1007/3-540-45061-0_62.
- 42 Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, October 1976. doi:10.1016/0304-3975(76)90061-X.
- 43 Klaus W. Wagner. More complicated questions about maxima and minima, and some closures of NP. *Theoretical Computer Science*, 51(1):53–80, January 1987. doi:10.1016/0304-3975(87)90049-1.
- 44 Klaus W. Wagner. Bounded query computations. *[1988] Proceedings. Structure in Complexity Theory Third Annual Conference*, pages 260–277, 1988. doi:10.1109/SCT.1988.5286.
- 45 James D. Watson and Johannes Bausch. The complexity of approximating critical points of quantum phase transitions, 2021. arXiv:2105.13350.
- 46 James D. Watson, Johannes Bausch, and Sevag Gharibian. The complexity of translationally invariant problems beyond ground state energies, 2020. arXiv:2012.12717.
- 47 James D. Watson and Toby S. Cubitt. Computational complexity of the ground state energy density problem, 2021. arXiv:2107.05060.