Electronic Colloquium on Computational Complexity, Report No. 149 (2021)

On polynomially many queries to NP or QMA oracles

Sevag Gharibian* and Dorian Rudolph*

November 3, 2021

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 $\mathrm{P}^{\mathrm{QMA}}$, 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 $\mathrm{P}^{\mathrm{NP}[\log]}$ and $\mathrm{P}^{\mathrm{QMA}[\log]}$, defined identically to P^{NP} and $\mathrm{P}^{\mathrm{QMA}}$, 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 $\mathrm{P}^{\mathrm{NP}[\log]}$.

In this work, we first show that for any verification class $C \in \{\mathrm{NP}, \mathrm{MA}, \mathrm{QCMA}, \mathrm{QMA}, \mathrm{QMA}(2), \mathrm{NEXP}, \mathrm{QMA}_{\mathrm{exp}}\}$, 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 $\mathrm{P}^{\mathrm{C}[\log]}$, and when $s \in O(\log^k(n))$, this yields bound $\mathrm{QP}^{C[\log^{k+1}]}$ (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.

1 Introduction

The celebrated Cook-Levin Theorem [Coo71; Lev73b] and Karp's 21 NP-complete problems [Kar72] 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 have been identified as NP-complete (see, e.g., [GJ79]). 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 [Sto76] defined the *Polynomial Hierarchy* (PH), of which the second level will interest us here. Specifically, one may consider $\Sigma_2^{\mathrm{P}} = \mathrm{NP}^{\mathrm{NP}}$ (i.e. an NP-machine with access to an NP-oracle) or $\Delta_2^{\mathrm{P}} = \mathrm{P}^{\mathrm{NP}}$ (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 [Kre92]: Given Boolean formula $\phi : \{0,1\}^n \mapsto \{0,1\}$, does the lexicographically largest satisfying assignment $x_1 \cdots x_n$ of ϕ have $x_n = 1$?

*Department of Computer Science and Institute for Photonic Quantum Systems (PhoQS), Paderborn University, Germany. Email: {sevag.gharibian, dorian.rudolph}@upb.de.

ISSN 1433-8092

Restricting the number of NP queries. In 1982, in pursuit of yet finer characterizations, Papadimitriou and Zachos [PZ82] 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 [Wag87; Wag88] 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 $\text{P}^{\|\text{NP}}$, then Hemachandra [Hem89] and Buss and Hay [BH91] have shown $\text{P}^{\|\text{NP}} = \text{P}^{\text{NP}[\log]}$. Thus, adaptivity appears crucial; so, Gottlob [Got95] 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 3.1). 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" [KSV02], 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 [Osb12; Boo14; Gha+15]).

The formalisation 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 [Wag87; Wag88] (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?) [Gha+15]. 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[1] 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!)

$\mathrm{P}^{\mathrm{QMA[log]}}$ versus $\mathrm{P}^{\mathrm{QMA}}$ and this paper. This sets up the question inspiring the current work — is $\mathrm{P}^{\mathrm{QMA[log]}} = \mathrm{P}^{\mathrm{QMA}}$? In 2020, Gharibian, Piddock, and Yirka [GPY20] showed that $\mathrm{P}^{\mathrm{QMA[log]}} = \mathrm{P}^{\|\mathrm{QMA}}$, for $\mathrm{P}^{\|\mathrm{QMA}}$ defined as $\mathrm{P}^{\|\mathrm{NP}}$ but with an NP-oracle. This gave a quantum analogue of $\mathrm{P}^{\|\mathrm{NP}} = \mathrm{P}^{\mathrm{NP[log]}}$ [Hem89; BH91], although it required completely different proof techniques[2]. In this paper, we thus set our sights on the next step: Gottlob's work on P^{NP} computations with trees as query graphs [Got95]. What we are able to achieve is not just a quantum analogue of [Got95], 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 $\mathrm{P}^{\mathrm{QMA[log]}}$, yielding a unified framework for mapping $\mathrm{P}^{\mathrm{QMA}}$-type problems directly to APX-SIM instances.

1.1 Our results

To state our results, define (formal definitions in Section 2)

$$\mathcal{QV} := \{\mathrm{NP}, \mathrm{MA}, \mathrm{QCMA}, \mathrm{QMA}, \mathrm{QMA(2)}, \mathrm{NEXP}, \mathrm{QMA}_{\mathrm{exp}}\}, \quad (1)$$

$$\mathcal{QV}^{+} := \mathcal{QV} \cup \{\mathrm{StoqMA}\}. \quad (2)$$

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[3] 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 3.1). As expected, deciding whether a given C-DAG corresponds to an accepting P^{C} computation is itself a P^{C}-complete problem (Lemma 3.6). 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 2.9). G has *separator number* [Gru12] $s(n)$ 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.

[1] If the definition of APX-SIM were to be modified so that the ground state energy of H was given as part of the input, then APX-SIM would be QMA-complete instead of $\mathrm{P}^{\mathrm{QMA}}$-complete. This is because once one knows the ground state energy, a single QMA query and no postprocessing suffices to answer APX-SIM.

[2] The roadblock quantumly is that unlike NP, QMA is a class of *promise problems*. Thus, one must account for the possibility that a (say) $\mathrm{P}^{\mathrm{QMA[log]}}$ machine may make "invalid" queries, i.e. those violating the promise of the QMA-oracle. A general survey covering such issues regarding promise problems is [O G06].

[3] Amplification here means that C with constant promise gap (difference between completeness and soundness parameters) is equal to C with $1/\operatorname{poly}$ gap.

Denote by C-DAG$_s$ a C-DAG of separator number s, where we write C-DAG$_1$ for the case of $s \in O(1)$. Note that treewidth upper bounds separator number [Gru12].

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 1.1. *Fix any $C \in \mathcal{QV}$ and efficiently computable function $s : \mathbb{N} \to \mathbb{N}$. Then,*

$$C\text{-DAG}_s \in \mathrm{DTIME}\left(2^{O(s(n)\log n)}\right)^{C[s(n)\log n]}, \tag{3}$$

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 1.1 in hand, we are able to obtain the following sequence of results.

First, by setting $s = O(1)$, we significantly strengthen Gottlob's [Got95] TREES(NP) $= \mathrm{P}^{\mathrm{NP}[\log]}$ result to the constant separator number case and broad range of verification classes C:

Theorem 1.2. *For any $C \in \mathcal{QV}$, C-DAG$_1$ is $\mathrm{P}^{C[\log]}$-complete.*

In words, any P^{C} computation with a query graph of constant separator number is decidable in $\mathrm{P}^{\mathrm{C}[\log]}$.

Second, an advantage of Theorem 1.1 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 1.3. *For all integers $k \geq 1$ and $C \in \mathcal{QV}$, C-DAG$_{\log^k} \in \mathrm{QP}^{C[\log^{k+1}(n)]}$, where QP denotes quasi-polynomial time (Definition 2.1).*

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 1.1 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., [AGL20]). Here, we also obtain non-trivial bounds, albeit weaker ones:

Theorem 1.4. *Fix $C = \mathrm{StoqMA}$ and any efficiently computable function $s : \mathbb{N} \to \mathbb{N}$. Then,*

$$C\text{-DAG}_s \in \mathrm{DTIME}\left(2^{O(s(n)\log^2 n)}\right)^{C[s(n)\log^2 n]}. \tag{4}$$

Note the extra log factor in the exponents — this prevents Theorem 1.4 from recovering result $\mathrm{P}^{\|\mathrm{StoqMA}} = \mathrm{P}^{\mathrm{StoqMA}[\log]}$ [GPY20] ($\mathrm{P}^{\|\mathrm{StoqMA}}$ corresponds to a StoqMA-DAG with $s(n) = 1$). Nevertheless, we *do* recover and improve on [GPY20] when we instead consider the case of bounded *depth* query graphs next.

Finally, Gottlob [Got95] also studied query graphs of bounded *depth*. The next theorem is an extension of his result. We define C-DAG$_d$ as C-DAG$_s$, except now we consider query DAGs of *depth* (Definition 4.5) at most d (as opposed to separator number s).

Theorem 1.5. *Let $d : \mathbb{N} \to \mathbb{N}$ be an efficiently computable function. For $C \in \{\mathrm{NP}, \mathrm{NEXP}, \mathrm{QMA}_{\mathrm{exp}}\}$, C-DAG$_d \subseteq \mathrm{P}^{C[d(n)\log(n)]}$, and for $C \in \mathcal{QV}^+$,*

$$C\text{-DAG}_d \subseteq \mathrm{DTIME}\left(2^{O(d(n)\log(n))}\right)^{C[d(n)\log(n)]}.$$

Using this, we obtain that deciding a P^{C} computation with a query graph of constant depth is $\mathrm{P}^{\mathrm{C[log]}}$-complete (Corollary 4.19). This modestly improves upon $\mathrm{P}^{\|\mathrm{StoqMA}} = \mathrm{P}^{\mathrm{StoqMA[log]}}$ [GPY20], which is the case of $d = 1$ (versus our $d \in O(1)$ in Theorem 1.5).

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 $\mathrm{P}^{\mathrm{QMA[log]}}$ problems) into APX-SIM: The "query gadget" construction of Ambainis [Amb14], and the "flag-qubit" framework[4] of Watson, Bausch, and Gharibian [WBG20] . 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-Hamiltonian[5] mapping used (and thus easily handles geometric constraints).

Here, we utilize the construction behind our main result, Theorem 1.1, 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 5.3) is beyond the scope of this introduction (below, we state a significantly simplified version[6]).

Lemma 1.6 ((Informal) Generalized Lifting Lemma (c.f. Lifting Lemma of [WBG20]))**.** *Fix $C \in \mathcal{QV}^+$ and any local circuit-to-Hamiltonian mapping H_{w} (Definition 5.2). 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 $C = \mathrm{StoqMA}$. Define $N := \min(N_s, N_d)$, and let G be a C-DAG instance n vertices of separator number $s(n)$ (as in Theorem 1.1) and depth $d(n)$ (as in Theorem 1.5). Then, there exists a $\mathrm{poly}(N)$-time many-one reduction from G to an instance (H, A) of APX-SIM, such that H has size $\mathrm{poly}(N)$ 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 $\mathrm{poly}(N)$ 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 [WBG20], which our Lemma 1.6 generalizes as follows: (1) [WBG20] requires parallel queries to C, whereas Lemma 1.6 allows arbitrary P^{C} computations (parameterized by separator number s), and (2) [WBG20] requires promise gap amplification for C, which is not known to hold for StoqMA, whereas Lemma 1.6 allows $C = \mathrm{StoqMA}$.

Next, by applying our lifting lemma for $C = \mathrm{QMA}$ and $s \in O(1)$, we obtain $\mathrm{P}^{\mathrm{QMA[log]}}$-hardness of APX-SIM (Theorem 5.7). This is not surprising, since our Theorem 1.2 shows C-DAG $\in \mathrm{P}^{\mathrm{QMA[log]}}$, and APX-SIM is $\mathrm{P}^{\mathrm{QMA[log]}}$-hard [Amb14; GY19]. What *is* interesting, however, is:

1. The map from P^{C} to APX-SIM of Lemma 1.6 is "direct", meaning we embed all the query dependencies of the input C-DAG directly into the flag qubit construction.
2. A poly-time reduction from $\mathrm{P}^{\mathrm{QMA}}$ to APX-SIM for all $1 \leq s \leq n$ would imply $\mathrm{P}^{\mathrm{QMA}} = \mathrm{P}^{\mathrm{QMA[log]}}$ and is therefore unlikely, if one believes $\mathrm{P}^{\mathrm{QMA}} \neq \mathrm{P}^{\mathrm{QMA[log]}}$. However, Lemma 5.3 shows $\mathrm{P}^{\mathrm{QMA}}$ *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 [WBG20] is somewhat mysterious, in that it "compresses" multiple QMA query answers into a *single* flag qubit, which *a priori* appears at odds

[4] This is a significantly generalized version of the "sifter" construction of Gharibian and Yirka [GY19].

[5] 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.

[6] For example, Lemma 5.3 also takes a separator tree as part of its input; for pedagogical purposes, the informal version presented here ignores this, as a separator tree is computed in $\mathrm{poly}(N)$ time in all our applications of Lemma 5.3 anyway.

with Holevo's theorem[7]. In the present paper, we reveal *why* this works — our construction utilizes the "admissible weighting function" framework of [Got95], 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 [WBG20] and here)!

In fact, we observe that [Amb14] implicitly rediscovers[8] a version of Gottlob's weighting function approach. Thus, underlying all three works of [Got95; Amb14; WBG20], as well as the current one, is a central unifying theme worth stressing:

Theme 1.7 (Unifying theme)**.** *The reduction of* P^{C} *to maximizing a single real-valued function.*

Finally, for $C = \mathrm{StoqMA}$ 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 5.8). 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 $\mathrm{P}^{\mathrm{StoqMA[log]}}$-hardness result of [GPY20] via the flag qubit framework (details in Section 5.4,) resolving an open question of [WBG20]. For clarity, [GPY20]'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 1.7, we close with a simple no-go statement purely about polynomials. Roughly, given a real-valued polynomial f (specified[9] via an arithmetic circuit (Definition 6.1)), 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 1.8. *Fix any function $h : R^+ \to R^+$. 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 6.2), p is weakly compressible to $h(B)$ bits. Then* $\mathrm{P}^{\mathrm{NP}} \subseteq \mathrm{P}^{\mathrm{NP}[h(B)]}$.

Let us be clear that this statement is not at all surprising for the reader familiar with Krentel's work [Kre88a] 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 1.8, one obtains:

Corollary 1.9. *If any multi-linear polynomial p (represented as an arithmetic circuit) can be weakly compressed with $h(B) = O(\log B)$, then* $\mathrm{P}^{\mathrm{NP}} \subseteq \mathrm{P}^{\mathrm{NP[log]}}$.

Corollary 1.10. *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* $\mathrm{P}^{\Sigma_2^p}$*).*

[7] Roughly, Holevo's theorem says that n qubits cannot reliably transmit more than n bits of information.

[8] Like [Got95], [Amb14] 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.

[9] Strictly speaking, we do not require arithmetic circuits to specify f. However, the multi-linear polynomials produced for our statement can have exponentially many terms if expanded fully in a monomial basis. To specify this succinctly, it suffices not to expand brackets in our polynomial descriptions (i.e. not replace $(x + y)(a + b) = xa + xb + ya + yb$); arithmetic cricuits are a natural avenue for formalising this.

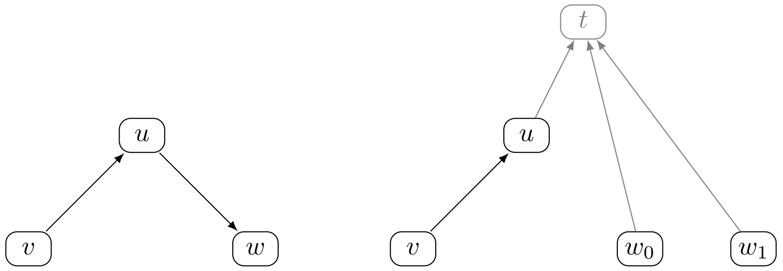


Figure 1: Simple example of a graph transformation, where the outputs of u are decoupled by creating copies w_0, w_1 with hardcoded inputs. t selects the copy of w depending on the output of u.

1.2 Techniques

1. Techniques for deciding C-DAGs. At a high-level, our approach follows Gottlob's strategy for P^{NP} [Got95]: Given[10] 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 [Got95] 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 [Got95] (rediscovered in [Amb14]), 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 4.3). 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 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" (Section 2.2.1) structure overlaying G. With this separator tree in hand, we show our main technical lemma, the Compression Lemma (Lemma 4.7). 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 [Got95]. 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

[10] Gottlob's modeling of query graphs is slightly different, in that nodes of the DAG encode propositional formulae, whereas here it is more convenient to put verification circuits at nodes.

and reroute wires, at the end of which we introduce a "conductor[11]" node t to orchestrate the madness. For the reader interested in a brief peek at details (Section 4.2), Figure 3 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 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 1.7); we use (Equation (25))

$$t(x, \psi_1, \ldots, \psi_N) := \sum_{i=1}^{N} f(v_i)\big(x_i \Pr[Q_i(z_i(x), \psi_i) = 1] + (1 - x_i)\gamma\big), \tag{5}$$

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 4.16), and (2) we can design a C-verifier V with acceptance probability precisely $t(x, \psi_1, \ldots, \psi_N)$ (up to renormalization) (Lemma 4.15). 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 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); this sleight of hand avoids complications regarding entanglement across proofs from previous works (e.g. [WBG20]).

2. Techniques for a unified APX-SIM framework. Roughly, [WBG20] embeds a (say) $\text{P}^{\text{QMA[log]}}$ 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 $H_{\text{w}}(V)$ 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 4.15). What we next show is that our V can rather simply be substituted for the superverifier V of [WBG20] in the flag qubit construction. The key reason this works is again Theme 1.7 — 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 [WBG20]). As in [WBG20], by then mapping V to $H_{\text{w}}(V)$, we can now penalize q on the Hamiltonian side to force all low energy states of $H_{\text{w}}(V)$ 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 [WBG20].

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

[11] Meant in the sense of an "orchestra conductor".

(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 (5) 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 1.10 leverages Hartmanis' result that if $\mathrm{P}^{\mathrm{NP}[2]} = \mathrm{P}^{\mathrm{NP}[1]}$, then $\mathrm{PH} = \mathrm{P}^{\Sigma_2^p}$ [Har93].

1.3 Related Work

The classes P^{NP} and $\mathrm{P}^{\mathrm{NP}[\log]}$. As mentioned above, $\mathrm{NP} \cup \mathrm{coNP} \subseteq \mathrm{P}^{\mathrm{NP}[\log]} \subseteq \Sigma_2^p$, and $\mathrm{P}^{\mathrm{NP}[\log]} \subseteq \mathrm{PP}$ [BHW89]. It holds that $\mathrm{P}^{\mathrm{NP}[\log]} = \mathrm{P}^{\|\mathrm{NP}}$ [Hem89; BH91]. Gottlob [Got95] showed that P^{NP} with a tree for a query graph equals $\mathrm{P}^{\mathrm{NP}[\log]}$ (this also follows from our Theorem 1.2). It is believed that for any $k \in O(1)$, the classes $\mathrm{P}^{\mathrm{NP}[k]}$, $\mathrm{P}^{\mathrm{NP}[\log^k n]}$, and P^{NP} are distinct. For example, $\mathrm{P}^{\mathrm{NP}[1]} = \mathrm{P}^{\mathrm{NP}[2]}$ implies both $\mathrm{P}^{\mathrm{NP}[1]} = \mathrm{P}^{\mathrm{NP}[\log]}$ and a collapse of PH to $\Delta_3^p = \mathrm{P}^{\Sigma_2^p}$ [Har93]. However, it is known that $\mathrm{P}^{\mathrm{NP}[\log^k(n)]} = \mathrm{P}^{\|\mathrm{NP}[\log^{k+1}(n)]}$ for all $k \geq 1$ [CS92]. Complete problems for $\mathrm{P}^{\mathrm{NP}[\log]}$ include determining a winner in Lewis Carroll's 1876 voting system [HHR97], and a $\mathrm{P}^{\mathrm{NP}[\log^2 n]}$-complete problem is model checking for certain branching-time temporal logics [Sch03].

Closely related to one of the central themes of this work, Theme 1.7, is Krentel's [Kre88b] work on OptP. Roughly, $\mathrm{OptP}[z(n)]$ 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 $\mathrm{OptP}[z(n)]$ and $\mathrm{FP}^{\mathrm{NP}[z(n)]}$ are equivalent (FP the set of *functions* computable in poly-time). Through this, [Kre88b] obtains (e.g.) that determining whether the length of the shortest traveling salesperson tour in a graph G is divisible by k is P^{NP}-complete, but that determining if the size of the max clique in G is divisible by k is only $\mathrm{P}^{\mathrm{NP}[\log]}$-complete. Before this, Papadimitriou had shown [Pap82] that deciding if G has a *unique* optimum traveling salesperson tour is P^{NP}-complete.

QMA, $\mathrm{P}^{\mathrm{QMA}[\log]}$ 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. [KR03; KKR06; D A+09; DS09]). For QMA(2), Chailloux and Sattath [CS12] showed the *separable sparse Hamiltonian problem* is QMA(2)-complete. Fefferman and Lin [FL16] prove that the local Hamiltonian problem with *exponentially* small promise gap is PSPACE-complete. See (e.g.) [Osb12; Gha+15] for surveys and further results.

Ambainis [Amb14] initiated the study of $\mathrm{P}^{\mathrm{QMA}[\log]}$, and showed APX-SIM is $\mathrm{P}^{\mathrm{QMA}[\log]}$-complete and SPECTRAL GAP (deciding if the spectral gap of a local Hamiltonian is large or small) is $\mathrm{P}^{\mathrm{UQMA}[\log]}$-hard. These results were obtained for log-local observables (APX-SIM) and Hamiltonians (APX-SIM and SPECTRAL GAP). Gharibian and Yirka [GY19] improve both results to $O(1)$-local, and show $\mathrm{P}^{\mathrm{QMA}[\log]} \subseteq \mathrm{PP}$. In contrast to $\mathrm{P}^{\mathrm{NP}[\log]}$, $\mathrm{P}^{\mathrm{QMA}[\log]}$ is *not* believed to be in PH, since even BQP is believed outside of PH [S A10; RT19]. Gharibian, Piddock, and Yirka [GPY20] next obtain a complexity classification of $\mathrm{P}^{\mathrm{QMA}[\log]}$ (along the lines of Cubitt and Montanaro [CM16]) depending on the class of Hamiltonians employed; this includes, for example, $\mathrm{P}^{\mathrm{StoqMA}[\log]}$-completeness for APX-SIM on stoquastic Hamiltonians. They also introduce the "sifter" framework to show the first $\mathrm{P}^{\mathrm{QMA}[\log]}$-hardness result for 1D Hamiltonians on the line. Watson, Bausch, and Gharibian [WBG20] 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 $\mathrm{P}^{\mathrm{QMA}_{\mathrm{exp}}}$-complete.

Most recently, Watson and Bausch [WB21] show a $\mathrm{P}^{\mathrm{QMA}_{\mathrm{exp}}}$-completeness result for approximating a critical boundary in the phase diagram of a translationally-invariant Hamiltonian. Aharonov and Irani [AI] and Watson and Cubitt [WC21] 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 $\mathrm{FP}^{\mathrm{NEXP}}$ and $\mathrm{FP}^{\mathrm{QMA}_{\mathrm{exp}}}$, while the latter shows that a decision version of the energy density problem is between $\mathrm{P}^{\mathrm{NEEXP}}$ and $\mathrm{EXP}^{\mathrm{QMA}_{\mathrm{exp}}}$ (for quantum Hamiltonians).

1.4 Open questions

First, can our main result (Theorem 1.1) be extended to further classes of graphs, perhaps by considering different parameterizations, such as graphs with logarithmic pathwidth (which includes the case of constant separator number)? Second, Theorem 1.1 gives non-trivial bounds when (say) $s \in O(1)$ or $s \in O(\mathrm{polylog}(n))$. For $s \in \Theta(n)$, however, the DTIME base therein scales as 2^n, thus yielding a trivial upper bound on C-DAG$_s$. Can our bound be improved from $\mathrm{DTIME}\left(2^{O(s(n)\log n)}\right)^{C[s(n)\log n]}$ to $\mathrm{DTIME}\left(2^{O(s(n))}\right)^{C[s(n)\log n]}$ (i.e. shave off the extra log factor in the base)? If so, one would immediately recover the $\mathrm{P}^{\|\mathrm{StoqMA}} \subseteq \mathrm{P}^{\mathrm{StoqMA}[\log]}$ result of [GPY20] (currently, we rely on Theorem 1.5 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 $\mathrm{DTIME}\left(2^{O(s(n))}\right)^{C[s(n)]}$ — since $\mathrm{P}^{\|\mathrm{NP}}$ has $s \in O(1)$, this would imply $\mathrm{P}^{\|\mathrm{NP}} = \mathrm{P}^{\mathrm{NP}[\log]} \in \mathrm{P}^{\mathrm{NP}[k]}$ 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 4.15 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

Notation. $S = \biguplus_i S_i$ denotes a partition of set S into subsets S_i. $:=$ denotes a definition.

Promise problems. Due to the inherently probabilistic nature of quantum computation, the quantum complexity classes we are interested in are defined in terms of *promise problems*. A promise problem Π is defined by a tuple $\Pi = (\Pi_{\mathrm{yes}}, \Pi_{\mathrm{no}}, \Pi_{\mathrm{inv}})$ with $\Pi_{\mathrm{yes}} \uplus \Pi_{\mathrm{no}} \uplus \Pi_{\mathrm{inv}} = \{0,1\}^*$. We call $x \in \Pi_{\mathrm{yes}}$ a *yes-instance*, $x \in \Pi_{\mathrm{no}}$ a *no-instance*, and $x \in \Pi_{\mathrm{inv}}$ an *invalid instance*.

Definition 2.1 (QP (quasi-polynomial time))**.** $\mathrm{QP} = \bigcup_k \mathrm{DTIME}(n^{\log^k n})$ is the set of problems accepted by a deterministic Turing machine in quasi-polynomial time.

2.1 Quantum Complexity Classes

The circuits used by quantum complexity classes belong to *polynomial-time uniform quantum circuit families* $\{Q_n\}$. That means, there exists a Turing machine that on input n outputs a classical description of a quantum circuit Q_n in time poly(n). Qubits are represented by the Hilbert space $\mathcal{B} := \mathbb{C}^2$.

The arguably most natural quantum analogue of NP (or MA) is QMA, where a BQP-verifier is given an additional quantum proof.

Definition 2.2 (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, $Q_n \in \mathcal{U}\left(\mathcal{B}_A^{\otimes n} \otimes \mathcal{B}_B^{\otimes p(n)} \otimes \mathcal{B}_C^{\otimes q(n)}\right)$. The register A is used for the input, B contains the proof, and C the ancillae initialized to $|0\rangle$.
- $\forall x \in \Pi_{\text{yes}}\ \exists |\psi\rangle \in \mathcal{B}^{\otimes p(|x|)} : \Pr[Q_{|x|} \text{ accepts } |x\rangle|\psi\rangle] \geq 2/3$
- $\forall x \in \Pi_{\text{no}}\ \forall |\psi\rangle \in \mathcal{B}^{\otimes p(|x|)} : \Pr[Q_{|x|} \text{ accepts } |x\rangle|\psi\rangle] \leq 1/3$

Here, we say a quantum circuit Q_n accepts an input $|x\rangle|\psi\rangle$ if measuring the first qubit of the ancilla register C in the standard basis results in outcome $|1\rangle$. The acceptance probability is then given by

$$\Pr[Q \text{ accepts } |x\rangle|\psi\rangle] = \left\| (I_A \otimes |1\rangle\langle 1|_{C_1} \otimes I)\, U |x\rangle_A |\psi\rangle_B |0\rangle_C^{\otimes q(n)} \right\|_2^2. \tag{6}$$

Note that the thresholds $c = 2/3$ and and $s = 1/3$ may be replaced with $c = 1 - \varepsilon$ and $s = \varepsilon$ such that $\varepsilon \geq 2^{-\operatorname{poly}(n)}$ [KSV02]. 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 $QMA(k)$, the verifier receives k unentangled proofs, i.e. $|\psi\rangle = \bigotimes_{j=1}^{k} |\psi_j\rangle)$. It holds that QMA(2) = QMA(poly(n)) as shown by Harrow and Montanaro [HM13]. 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.

The classical complexity classes NP and MA (Merlin-Arthur) may also be considered special cases of QMA. Restricting QMA to classical proofs and classical randomized verifiers results in MA. Additionally requiring perfect completeness and soundness yields NP. Note that NP and MA are usually equivalently defined as the problems accepted by nondeterministic (randomized) Turing machines.

Next, we define the k-local Hamiltonian problem, which was shown to be QMA-complete in a "quantum Cook-Levin theorem" by Kitaev [KSV02].

Definition 2.3 (k-local Hamiltonian)**.** A Hermitian operator $H \in \operatorname{Herm}(\mathcal{B}^{\otimes n})$ acting on n qubits is a k-local Hamiltonian if it can be written as

$$H = \sum_{\substack{S \subseteq [n] \\ |S| \leq k}} H_S \otimes I_{[n]\setminus S}. \tag{7}$$

Additionally, $0 \preccurlyeq H_S \preccurlyeq I$ 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 2.4 (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 $\lambda_{\min}(H) \geq b$.

Next, we give formally define the oracle based complexity classes used throughout this paper.

Definition 2.5 (P^C)**.** Let C be a complexity class with complete problem Π. $\text{P}^C = \text{P}^\Pi$ 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 $\Gamma \in \mathrm{P}^C$ 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 $\mathrm{P}^{C[f]}$ 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 $\mathrm{P}^{C[k]}$, where M may ask at most k queries on each input.

$\mathrm{P}^{\|C}$ 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.

For a function $f : \{0,1\}^* \to \{0,1\}^*$, we define P^f and the other classes analogously, except that M may now query the oracle for values $f(x)$.

The $\mathrm{P}^{\mathrm{QMA[log]}}$-complete problem is APX-SIM (approximate simulation). It essentially asks whether a given Hamiltonian has a ground state with a certain property (e.g., a ground state where the first qubit is set to $|1\rangle$).

Definition 2.6 (APX-SIM$(H, A, k, l, a, b, \delta)$ [Amb14])**.** 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 $\langle\psi|A|\psi\rangle \geq b$.

Ambainis showed completeness for $k = \Theta(\log n)$ [Amb14]. Gharibian and Yirka [GY19] improved this to $k = 5$. Gharibian, Piddock, and Yirka [GPY20] improved this to $k = 2$ for physically motivated Hamiltonian models.

Definition 2.7 (StoqMA [BBT06a])**.** Fix polynomials $\alpha(n), \beta(n), p(n), q(n), r(n)$ with $\alpha(n) - \beta(n) \geq 1/\operatorname{poly}(n)$. A promise problem Π is in StoqMA (Stoquastic Merlin Arthur) if there exists a polynomial-time uniform quantum circuit family $\{Q_n\}$ such that the following holds:

- For all n, $Q_n \in \mathcal{U}\left(\mathcal{B}_A^{\otimes n} \otimes \mathcal{B}_B^{\otimes p(n)} \otimes \mathcal{B}_C^{\otimes q(n)} \otimes \mathcal{B}_D^{\otimes r(n)}\right)$. The register A is used for the input, B contains the proof, C ancillae initialized to $|0\rangle$, and D ancillae initialized to $|+\rangle$. Q_n only uses X, CNOT, and Toffoli gates.
- For $x \in \{0,1\}^*$, $|x| = n$, $|\psi\rangle \in \mathcal{B}^{\otimes p(n)}$, let $|\psi_{\text{in}}\rangle := |x\rangle_A|\psi\rangle_B|0\rangle_C^{\otimes q(n)}|+\rangle_D^{\otimes r(n)}$. The acceptance probability is then given by
$$\Pr[Q_n \text{ accepts } |x\rangle|\psi\rangle] = \langle\psi_{\text{in}}|Q_n^\dagger \Pi_{\text{acc}}\, Q_n|\psi_{\text{in}}\rangle, \tag{8}$$
where $\Pi_{\text{acc}} = |+\rangle\langle+|_{C_1}$ measures the first ancilla in the $\{|+\rangle, |-\rangle\}$ basis.
- $\forall x \in \Pi_{\text{yes}}\ \exists|\psi\rangle \in \mathcal{B}^{\otimes p(|x|)} : \Pr[Q_{|x|} \text{ accepts } |x\rangle|\psi\rangle] \geq \alpha(n)$
- $\forall x \in \Pi_{\text{no}}\ \forall|\psi\rangle \in \mathcal{B}^{\otimes p(|x|)} : \Pr[Q_{|x|} \text{ accepts } |x\rangle|\psi\rangle] \leq \beta(n)$

Note that the only difference between StoqMA and MA is that StoqMA may perform its final measurement in the $\{|+\rangle, |-\rangle\}$ basis (i.e. setting $\Pi_{\text{acc}} := |0\rangle\langle 0|_{C_1}$ would result in MA) [BBT06a].

It further holds that the StoqMA verifier accepts any state with only nonnegative coordinates with probability $\geq 1/2$. Therefore, we cannot amplify the gap by majority voting as for MA. Recently, Aharonov, Grilo, and Liu [AGL20] have shown that StoqMA with $\alpha(n) = 1 - \operatorname{negl}(n)$ and $\beta(n) = 1 - 1/\operatorname{poly}(n)$ is contained in MA, where $\operatorname{negl}(n)$ denotes a function smaller than all inverse polynomials for sufficiently large n. It is therefore unlikely that such an amplification is possible.

2.2 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 2.8 (Tree decomposition)**.** Let $G = (V, E)$ be an undirected graph. A *tree decomposition* $T = (V_T, E_T)$ of G is a graph with m nodes labelled by subsets $X_1, \dots, X_m \subseteq V$ such that:

- Each node of G is contained in some node X_i of T: $\bigcup_{i=1}^m X_i = V$.
- For all $(u, v) \in E$, there exists an X_i such that $u, v \in X_i$.
- For all $v \in V$, the subtree in T induced by $\{X_i \mid v \in X_i\}$ is connected.

The *width* of a tree decomposition T is defined as $\text{width}(T) := \max_i |X_i| - 1$. The *treewidth* of G, denoted $\text{tw}(G)$, is defined as the minimum width among all possible tree decompositions of G.

Bodlaender [Bod93] has shown that tree decompositions for graphs with bounded treewidth (i.e. $\text{tw}(G) = O(1)$) can be computed in linear time.

The connection between tree decompositions and *separators*, which we define next, has a long and well-studied history (e.g. [RS86; Ree92; Bod+95; Ami10; Bod+13]).

Definition 2.9 (Separator number [Gru12])**.** 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 $\text{s}(G)$, 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 2.10 (Theorem 9 of [Gru12] (see also[12] [RS86; Bod+95]))**.** $\text{s}(G) \leq \text{tw}(G) \leq O(s(G) \cdot \log n)$.

We define tree decompositions and separator number for a directed graph G on the undirected version of G. It appears to be an open problem whether $\text{tw}(G) = \Theta(s(G))$ holds. However, resolving this question would not improve our results, since we only use the first inequality.

2.2.1 Separator Trees

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

Definition 2.11. 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 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 $G \setminus S_1$. To distinguish vertices/edges of G from vertices/edges of T, we refer to the latter 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.

Lemma 2.12. *Given an n-vertex graph $G = (V, E)$, a separator tree T of G with separator number $s := \text{s}(G)$ can be computed in time $n^{O(s)}$.*

[12] Proposition 2.5 of [RS86] gives the slightly weaker bound $\text{s}(G) \leq \text{tw}(G) + 1$, which also suffices for our purposes.

Proof. By Definition 2.9, every induced subgraph of G has a balanced separator of size at most s. Thus, the brute force approach to build a separator tree is to first brute force search for a separator S of G in time $n^{s'}$ for $s' = s+2$ (try all $\binom{n}{s}$ subsets of vertices, for each subset check $O(n^2)$ edges), remove it, and recurse on all induced balanced subgraphs on $V \setminus S$. (Technically, since we do not know s beforehand, we can try all values for separator size starting from 2 onwards via brute force; this does not affect the overall runtime.)

To analyze the runtime of this procedure over all recursive calls, a slight non-triviality is that for a balanced separator S (Definition 2.9), we have no control over the sizes of each connected component of $G \setminus S$, other than no one component has size more that $|V|/2$. Thus, the recurrence relation one obtains scales as $T(n) = \left(\sum_{i=1}^{k} T(n_i)\right) + n^{s'}$, where $2 \leq k \leq n$, $\sum_{i=1}^{k} n_i \leq n$, $n_i \leq n/2$ for all i, and for some $s' = s+2$. (In particular, this means the standard Master Theorem [BHS80] cannot be applied.) In fact, the values of the n_i can even change between levels of the recurrence.

The analysis, luckily, is simple. Let $L = 1$ denote the base case of the recurrence, which we view as the root of a recursion tree (i.e. each node v of the tree is a recursive call, whose children correspond to the recursive calls made by v). At any level $L \geq 1$, we claim the additive cost at a node v (i.e. corresponding to the "$+n^{s'}$" term) is at least twice the additive cost of its children. This implies the total cost incurred at level $L+1$ is at most half the cost of level L, giving a total cost for the algorithm via geometric series $\sum_{L=0}^{D-1} \frac{n^{s'}}{2^L} \leq 2n^{s'}$, for all D denoting the depth of the recursion, as claimed.

To see that the cost at any v is indeed at least twice the cost of its children $w_1, \ldots, w_k$, let n be the input size for v and $n_1, \ldots, n_k$ the input sizes for $w_1, \ldots, w_k$, respectively. Then, the total additive cost across all children of v is

$$\sum_{i=1}^{k} n_i^{s'} = n^{s'} \sum_{i=1}^{k} \frac{n_i}{n} \left(\frac{n_i}{n}\right)^{s'-1} \leq n^{s'} \max_i \left(\frac{n_i}{n}\right)^{s'-1} \leq \frac{1}{2} n^{s'}, \tag{9}$$

where the first inequality follows since the coefficients n_i/n yield a convex combination, and the second inequality since $n_i \leq n/2$ for all i and $s' = s+2 \geq 1$. □

Remark 2.13. Note that the separator tree computed by Lemma 2.12 may contain separators of different sizes $1 \leq s' \leq s$. However, in this work it is convenient to assume without loss of generality that all separators have size exactly s. This can trivially be achieved by "padding" each separator S of size $1 \leq s' < s$ by adding dummy vertices to S (and hence to G; all dummy vertices are isolated). The number of dummy vertices added is trivially at most sn (there can never be more than n separators); thus, the size of G increases by at most sn vertices, which does not affect any of our results.

Additionally, although by definition of balanced separator, 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.

Lemma 2.14. *Given an n-vertex graph G, depth D and separator size s (D and s are specified in unary), a separator tree of G of depth D with separators of size s can be computed in time $n^{O(Ds)}$, if it exists.*

Proof. A brute-force approach similar to Lemma 2.12 is used, except there is a catch: At any level L of the recursion, for each subset of vertices S we consider, even if S is a separator, it may not lead to a *depth D* separator tree, even if such a tree exists. Thus, it does not suffice at level L to simply find a size s separator S, but rather in the worst case we may need to consider all such $O(n^s)$ such separators S. Thus, the recurrence relation now scales as $T(n) = n^s \left(\sum_{i=1}^{k} T(n_i)\right) + n^{s'}$. Running the same argument as Lemma 2.12 now yields total cost $\sum_{L=0}^{D-1} n^{s'} \left(\frac{n^s}{2}\right)^L \in n^{O(Ds)}$, where recall $s' = s+2$. □

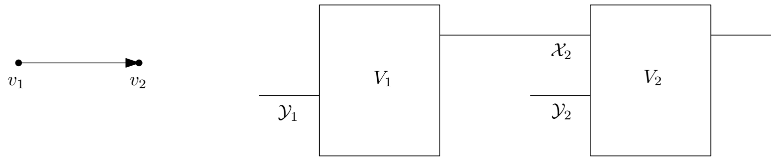


Figure 2: Left: A simple example of an NP-DAG for with two nodes, with v_2 the output node. Right: The circuit view represented by the NP-DAG. Each V_i is an NP verifier taking in input in register $\mathcal{X}_i$ and proof in register $\mathcal{Y}_i$. Note v_1 has in-degree 0, hence V_1 has trivial input register $\mathcal{X}_1$. The output wire of V_2 carries the output of the NP-DAG.

Algorithm 1 Evaluation procedure for C-DAG

```
function EVALUATE($G = (V, E)$)
    Sort the nodes of $V$ topologically into $v_1, \ldots, v_n$.
    The variable $x_i \in \{0, 1\}$ will denote the result of $v_i$'s query.
    for $i = 1, \ldots, n$ do
        $z_i \leftarrow \bigcirc_{v_j \in \text{parents}(v_i)} x_j$    ▷ $\bigcirc$ denotes concatenation (concatenation order is specified by $Q_i$).
        $x_i \leftarrow \begin{cases} 1, & \text{if } z_i \in \Pi^i_{\text{yes}} \\ 0, & \text{if } z_i \in \Pi^i_{\text{no}} \\ 0 \text{ or } 1 \text{ (nondeterministically)}, & \text{if } z_i \in \Pi^i_{\text{inv}} \end{cases}$
    return $x_n$    ▷ Recall $v_n$ is the result node.
```

Finally, we remark that only the size s of the separators in the balanced (or low-depth) separator tree is relevant for our algorithms. The separator number $\mathrm{s}(G)$ is only used to compute separator trees more efficiently. For a balanced separator tree, we may have $\mathrm{s}(G) \geq \Omega(s \cdot \log(n))$.[13]

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 3.1 (C-DAG (Figure 2))**.** Fix any complexity class $C \in \mathcal{QV}^+$. A C-DAG instance is defined by an n-node DAG $G = (V = \{v_1, \ldots, v_n\}, E)$, 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 poly(n)-sized description[14] of a verification circuit Q_i with designated input and proof registers $\mathcal{X}_i$ and $\mathcal{Y}_i$.[15] 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 poly(n) bits. v_i has a single output wire, denoted 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).

[13] Proof: Let G be a complete binary tree on n-nodes with additional edges from each node to its descendants. Then $\mathrm{s}(G) = \Theta(\log n)$, but G has a separator tree with separators of size 1.

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

[15] For example, if $C = \text{NP}$, then Π^i_{yes} 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).

Remark 3.2. Observe that if C is a promise class, then C-DAG is a *promise* problem (as opposed to a decision problem) — this is because then Π^i_{inv} is not necessarily empty, and so we must be *promised* that Algorithm 1 outputs either 0 or 1 deterministically, regardless of how invalid queries are answered.

Definition 3.3 (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 3.4. Intuitively, in Definition 3.3 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). Also, we may view any query string as a function $V \to \{0,1\}$.

Remark 3.5. Our notion of C-DAG is similar to the DAGS(NP) formalization of Gottlob (Definition 3.2 of [Got95]), except the latter has node queries encoded by propositional formulae. In contrast, here we use verification circuits at the nodes to make it easier to abstractly address a variety of verification classes C. (Alternatively, one might also consider "quantizing" the NP-dags of [Got95] by replacing propositional formulae with local Hamiltonians.)

Just as Gottlob shows DAGS(NP) (more accurately, DAGS(SAT)) is P^{NP}-complete [Got95], here we have the more general statement:

Lemma 3.6. *For any $C \in \mathcal{QV}^+$, C-DAG is P^C-complete.*

Proof. First, C-DAG $\in \text{P}^C$ holds because a P^C machine can straightforwardly compute a correct query string by simulating Algorithm 1 on a C-DAG-instance G. By definition, if $G \in C\text{-DAG}_{\text{yes}}$, then $x_n = 1$, and if $G \in C\text{-DAG}_{\text{no}}$, then $x_n = 0$.

Second, to show P^C-hardness of C-DAG, we sketch a poly-time many-one reduction from P^C to C-DAG. Let M be a P^C machine receiving input $x \in \{0,1\}^*$. Without loss of generality, we may assume that M always performs $m \leq \text{poly}(|x|)$ queries, so let G be a DAG with m nodes. Node v_i represents the ith query of M and has incoming edges from $v_1, \ldots, v_{i-1}$ (i.e. query i depends on all previous queries). Then, Q_i is defined as the circuit that, conditioned on the answers of queries 1 through $i-1$, first computes the C-query ϕ (e.g. ϕ could be a SAT formula or a local Hamiltonian) which M would send to the C-oracle for query i, and simulates the corresponding C-verification circuit on ϕ, outputting the result of said verification. (For clarity, note the C-verifier is not actually "run" here; we are simply defining the action of Q_i as part of the query graph for the reduction.) By construction and how YES/NO instances of C-DAG are defined (Algorithm 1), $G \in C$-DAG if and only if M accepts x. □

Remark 3.7. When C is a promise class, P^C is also a promise class (despite having P as a base). This is because, as with the definition of C-DAG (Definition 3.1), a valid P^C machine is promised to deterministically output the same answer regardless of how invalid queries are answered.

Thus, Lemma 3.6 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 2.10, includes the case of bounded treewidth). For this, we introduce the following definition for convenience.

Definition 3.8 ($C\text{-DAG}_s$)**.** Let $s : \mathbb{N} \to \mathbb{N}$ be an efficiently computable function. Then, $C\text{-DAG}_s$ 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 $C\text{-DAG}_1$ to denote the case of $s \in O(1)$.

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

4 Query Graphs with Bounded Separator Number

We first state the main technical theorem of this section, Theorem 4.1, followed by the results we obtain from it as corollaries. The remainder of Section 4 then proves Theorem 4.1. 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.

Theorem 4.1. *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 4.2. The class StoqMA is not included in Theorem 4.1; this is because the proof of the theorem requires C with a constant promise gap, which StoqMA is not known to have. (See Section 4.4 for the weaker result we are able to show for StoqMA.)

With Theorem 4.1 in hand, we obtain the following results.

Theorem 1.1. *Fix any $C \in \mathcal{QV}$ and efficiently computable function $s : \mathbb{N} \to \mathbb{N}$. Then,*

$$C\text{-DAG}_s \in \text{DTIME}\left(2^{O(s(n)\log n)}\right)^{C[s(n)\log n]}, \tag{3}$$

for n the number of nodes in G.

Proof. A separator tree of depth $D = O(\log(n))$ and separators of size $s = \mathrm{s}(G)$ is computed using Lemma 2.12 in time $n^{O(s)}$. Applying Theorem 4.1 completes the proof. □

In words, this says that P^C, with the restriction that the query graph used by the P machine has separator number $f(n)$, is contained in the class on the right side of Equation (3). When $f \in O(1)$, this upper bound is tight:

Theorem 1.2. *For any $C \in \mathcal{QV}$, C-DAG$_1$ is $\mathrm{P}^{C[\log]}$-complete.*

Proof. C-DAG$_1 \in \mathrm{P}^{C[\log]}$ is immediate from Theorem 1.1. As for $\mathrm{P}^{C[\log]}$-hardness, we use the well-known fact that $\mathrm{P}^{C[\log]} \subseteq \mathrm{P}^{\|C}$ for general [Bei91][16] C, and observe $\mathrm{P}^{\|C}$-hardness of C-DAG$_1$. Namely, the DAG G for any input to a problem from $\mathrm{P}^{\|C}$ is a star, with all edges directed towards the center of the star, which is the output node. Thus, G has separator size 1 (i.e. remove the center of the star to isolate all remaining vertices), i.e. it encodes an instance of C-DAG$_1$. □

More generally, we obtain the following general scaling corollary when the separator number is polylogarithmic.

Corollary 1.3. *For all integers $k \geq 1$ and $C \in \mathcal{QV}$, C-DAG$_{\log^k} \in \mathrm{QP}^{C[\log^{k+1}(n)]}$, where QP denotes quasi-polynomial time (Definition 2.1).*

Organization of remainder of section. Section 4.1 introduces the notion of weighting functions. Section 4.2 gives the main graph transformation which "compresses" a C-DAG instance appropriately, and sets up a corresponding weighting function. This can be roughly thought of as a "hardness proof", i.e. that the compressed DAG output by this graph transformation captures the original C-DAG instance.

[16] Reference [Bei91] actually studies only NP, but the containment proof technique straightforwardly generalizes to other classes: Namely, instead of making logarithmically many adaptive queries to C, precompute the polynomially many potential queries the P machine could make, and send these in one parallel round to the C-oracle.

Section 4.3 gives the matching upper bound — that the compressed DAG, coupled with an appropriate choice of weighting function, can now be resolved with fewer queries to a C-oracle. Section 4.4 and Section 4.5 discuss the special cases of StoqMA and bounded depth (beyond the naive log bound)) C-DAG instances, respectively.

Notation. The remainder of this section introduces a fair amount of notation. For ease of reference, we collect notation here. $\Gamma(v) := \{w \mid (v,w) \in E\}$ is the neighbor set of vertex v. The descendents of vertex v are denoted $\mathrm{Desc}(v)$, 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 $\mathrm{Anc}(v)$, 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 [Got95] to prove TREES(NP) = $\mathrm{P}^{\mathrm{NP[log]}}$, and later implicitly by Ambainis [Amb14] to show $\mathrm{P}^{\mathrm{QMA[log]}}$-hardness of the Approximate Simulation (APX-SIM) problem. We use a modified definition.

Definition 4.3 (Weighting function)**.** Let $G = (V, E)$ be a DAG. An efficiently computable function $f : V \to \mathbb{R}$ is called a *weighting function.* We say f is *c-admissible* for constant $c \in \mathbb{R}$ if for all $v \in V$,

$$f(v) \geq 1 + c \sum_{w \in \Gamma(v)} f(w), \tag{10}$$

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

$$W_f(G) = \sum_{v \in V} f(v). \tag{11}$$

Remark 4.4. Our Definition 4.3 is slightly weaker than Gottlob's [Got95], which sums over all nodes in $\mathrm{Desc}(v)$ (i.e. nodes reachable from v via a directed path, excluding v itself) instead of $\Gamma(v)$ in (10). However, these definitions are equivalent up to a constant factor in c.

Definition 4.5 (Levels of a DAG)**.** Let $G = (V, E)$ be a DAG. We divide G recursively into levels. Level 0 is made up by the nodes without incoming edges. Level $i + 1$ contains nodes v that have only inputs w (i.e. $(w, v) \in E$) with $\mathrm{level}(w) \leq i$ and at least one input w with $\mathrm{level}(w) = i$. We denote the level of a node $v \in V$ by $\mathrm{level}(v)$. Nodes on the last level are called *terminal nodes.* The *depth* of G, denoted $\mathrm{depth}(G)$ is the maximum level number.

In the next lemma, we extend Gottlob's [Got95] admissible weighting functions to our definition of c-admissability (Definition 4.3). For $c = 1$, the definitions are the same.

Lemma 4.6. *For any DAG $G = (V, E)$ and $c \geq 2$, the weighting functions ρ and ω below are c-admissible:*

$$\rho(v) = (c|V|)^{\mathrm{depth}(G) - \mathrm{level}(v)} \tag{12}$$

$$\omega(v) = (c+1)^{|\mathrm{Desc}(v)|} \tag{13}$$

Proof. The proof for ρ is the same as in [Got95], whereas our proof for ω is significantly simplified. To argue c-admissability of ρ, let $v \in V$. By Definition 4.5, it holds that $\mathrm{level}(w) > \mathrm{level}(v)$ for all $w \in \Gamma(v)$.

Therefore,

$$1 + c \sum_{w \in \Gamma(v)} \rho(w) \leq c|V|(c|V|)^{\mathrm{depth}(G)-\mathrm{level}(v)-1} = (c|V|)^{\mathrm{depth}(G)-\mathrm{level}(v)} = \rho(v). \tag{14}$$

For ω, let $\{u_1, \ldots, u_k\} = \Gamma(v)$ be topologically ordered (with respect to G). Then $|\mathrm{Desc}(u_i)| \leq |\mathrm{Desc}(v)| - i$. Thus,

$$\begin{aligned}
1 + c\sum_{i=1}^{k} \omega(u_i) &\leq 1 + c\sum_{i=1}^{k}(c+1)^{|\mathrm{Desc}(v)|-i} && (15)\\
&\leq 1 + c \sum_{i=0}^{|\mathrm{Desc}(v)|-1} (c+1)^i && (16)\\
&= 1 + c\frac{(c+1)^{|\mathrm{Desc}(v)|} - 1}{c} && (17)\\
&= (c+1)^{|\mathrm{Desc}(v)|} && (18)\\
&= \omega(v). && (19)
\end{aligned}$$

□

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 4.7. *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* Evaluate(G) = Evaluate(G^*) *(irrespective of nondeterministic choices in Algorithm 1). As required by the definition of C-DAG (Definition 3.1), each node of G^* corresponds to a verification circuit of size* poly$(|V^*|)$.

Combining this with Section 4.3, we will hence be able to decide G^* with $O(sD)$ queries.

Brief outline. The transformation from G to G^* proceeds in multiple steps. First, we construct a graph G' (Section 4.2.1) where each node $v \in V'$ has $|\mathrm{Desc}(v)| \leq O(sD)$, where recall $\mathrm{Desc}(v)$ is the set of descendents of v. Roughly, this is achieved by exploiting the structure of separator trees to "hardcode" dependencies. This leaves two issues. First, for technical reasons G' is lacking an output node, which we add in G''. Second, we have redundant copies of nodes, which simplify the construction, but are problematic in the presence of invalid queries, as two copies of the same node with the same inputs may produce different outputs. We merge these redundant nodes to obtain graph G^*, and define a suitable c-admissible weighting function in the process (Section 4.2.2). Section 4.2.3 shows correctness.

4.2.1 Basic Construction (G')

In this section, we construct $G' = (V', E')$. We begin by formally stating the construction, followed by giving the intuition, and an illustration via Figure 3b and accompanying discussion.

The graph transformation. Let $T = (V_T, E_T)$ be a separator tree (Definition 2.11) of G of depth D and separator size s. A running example is given in Figure 3a. Let $S \in V_T$ be an arbitrary supervertex and $S_1, \ldots, S_d$ 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 by some subset of s vertices, $S = (u_{S,1}, \ldots, u_{S,s})$, 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

$$V_S := \left\{ v_{S,i}^{z_1,\ldots,z_d} \;\middle|\; i \in [s],\, z_1, \ldots, z_d \in \{0,1\}^s \right\} \tag{20}$$

and set $V' = \bigcup_{S \in V_T} V_S$. As depicted in Figure 3b, 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, $v_{S,i}^{z_1,\ldots,z_d}$ in V' represents node $u_{S,i}$ in V, but conditioned on "outcome strings" $z_1, \ldots, z_d \in \{0,1\}^s$ in the separators $S_1, \ldots, S_d$. For ease of reference, we define a surjective function preimage : $V' \mapsto V$ to formalize this relationship:

$$\forall S, i, z_1, \ldots, z_d, \quad \text{preimage}(v_{S,i}^{z_1,\ldots,z_d}) = u_{S,i}. \tag{21}$$

Finally, since T has at most n supernodes, we have $|V'| \leq 2^{O(sD)} n$.

Next, define edges

$$E_S = \left\{ \left(v_{S,i}^{z_1,\ldots,z_d}, v_{S_j,k}^{z_1,\ldots,z_j}\right) \;\middle|\; i \in [s],\, j \in [d-1],\, u_{S_j,k} \in \text{Desc}(u_{S,i}) \right\}, \tag{22}$$

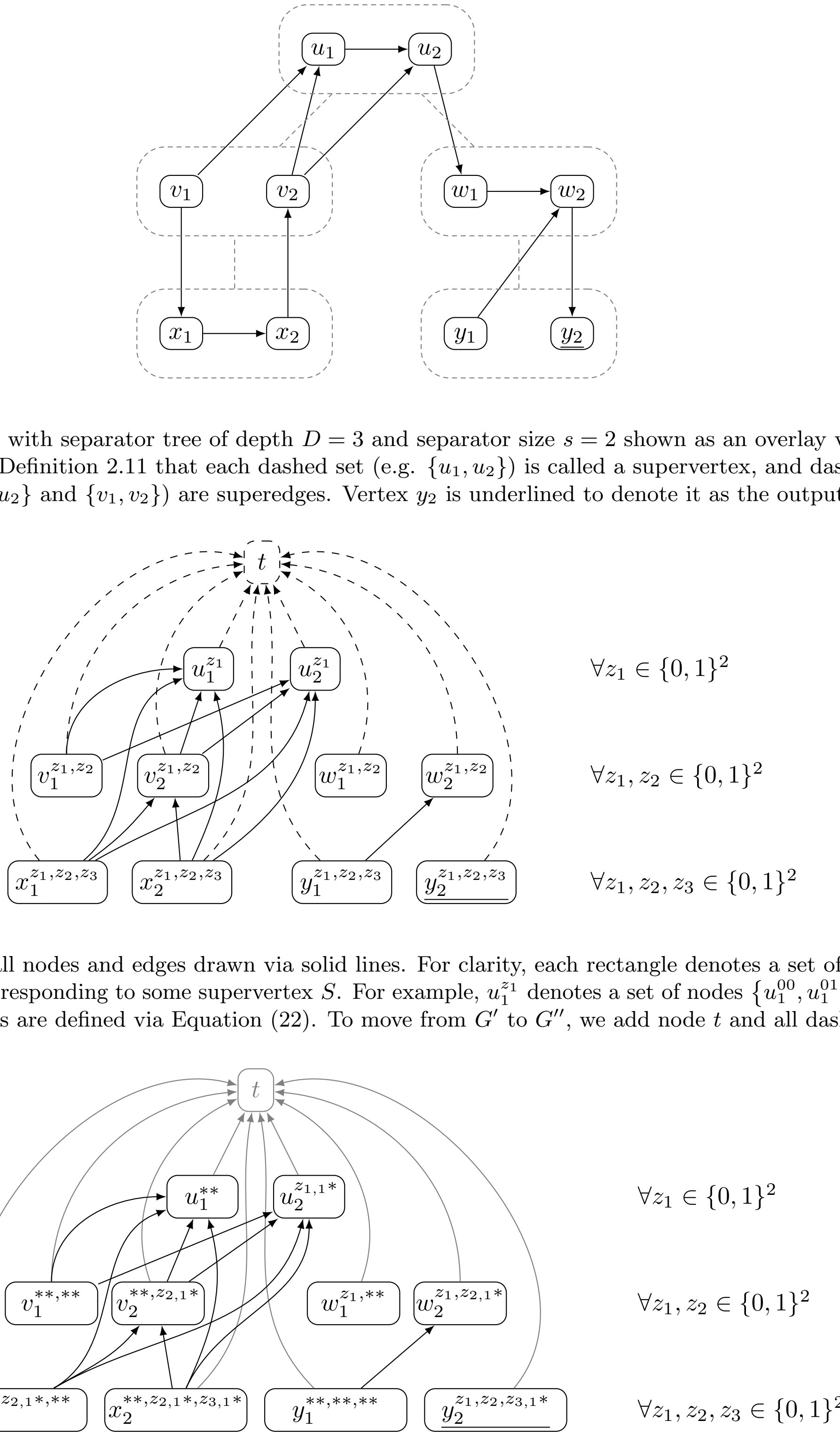
where recall $u_{S_j,k} \in \text{Desc}(u_{S,i})$ is the set of all descendants of $u_{S,i}$ in the original graph G. In words, each E_S creates, for each copy of $u_{S,i}$, edges to all copies of descendants u_{S_j} which are on a strictly higher level in the separator tree (due to the $j \in [d-1]$ constraint). In the context of Figure 3a, this means we "shortcut" paths to descendents, 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 $G \mapsto G'$, 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 $v_{S,i}^{z_1,\ldots,z_d} \in V'$. Roughly, the goal is for the query at $v_{S,i}^{z_1,\ldots,z_d}$ to simulate the query at preimage$(v_{S,i}^{z_1,\ldots,z_d}) = u_{S,i}$. However, $v_{S,i}^{z_1,\ldots,z_d}$ is "conditioned" on bit strings $z_1, \ldots, z_d$, so the simulation is not straightforward. To make this formal, we use Algorithm 2 as follows:

Rule 4.8. For each edge $(u_{T,j}, u_{S,i})$ in G, the result of $\textsc{ComputeOutput}(u_{T,j} \mid z_1, \ldots, z_d)$ is used as the corresponding input to $v_{S,i}^{z_1,\ldots,z_d} \in V'$.

Intuitively, we may view the conditioning string $z_1, \ldots, z_d$ 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 the $\textsc{ComputeOuput}()$ function.

Remark 4.9. (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 $z_1, \ldots, z_d$ corresponding



(a) Query graph G with separator tree of depth $D = 3$ and separator size $s = 2$ shown as an overlay via dashed lines. Recall from Definition 2.11 that each dashed set (e.g. $\{u_1, u_2\}$) is called a supervertex, and dashed edges (e.g. between $\{u_1, u_2\}$ and $\{v_1, v_2\}$) are superedges. Vertex y_2 is underlined to denote it as the output node.

(b) G' consists of all nodes and edges drawn via solid lines. For clarity, each rectangle denotes a set of nodes V_S (Equation (20)) corresponding to some supervertex S. For example, $u_1^{z_1}$ denotes a set of nodes $\{u_1^{00}, u_1^{01}, u_1^{10}, u_1^{11}\}$, whose neighbor sets are defined via Equation (22). To move from G' to G'', we add node t and all dashed edges.

(c) Graph G^* with merged nodes indicated by asterisks in the superscript.

Figure 3: Example of the query graph transformation.

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

```
1: function ComputeOutput($u_{S,i} \mid z_1, \ldots, z_m$)                    ▷ recall $u_{S,i} \in S$
2:     $S_1, \ldots, S_d \leftarrow$ 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 $j = 1, \ldots, s$ do          ▷ in topological order, set answer bits at current level of recursion
7:         $z_{m+1,j} \leftarrow$ out-wire $\left[u^{z_1,\ldots,z_{m+1}}_{S_{m+1},j}\right]$   ▷ set bit $j$ of $z_{m+1}$ using query answers on incoming edges
8:     return ComputeOutput($u_{S,i} \mid z_1, \ldots, z_{m+1}$)
```

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 $v^{z_1,\ldots,z_d}_{S,i}$ in the same branch. For this, $v^{z_1,\ldots,z_d}_{S,i}$ will need access to the out-wire functions (Definition 3.1) of certain nodes below it; this is afforded to $v^{z_1,\ldots,z_d}_{S,i}$ via the edge set E_S (demonstrated via the "upward" black edges in Figure 3b).

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 3b 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'. To solve both problems simultaneously, 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 ComputeOutput($v \mid \varepsilon$) 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 3b via dashed lines.

Intuition. The construction of G'', and why it does what we need, is subtle; so let us illustrate via Figure 3b. For this, recall in Figure 3a that $y_2 \in V$ is the original output node of G. For concreteness, assume in this discussion that $C = \text{NP}$ and $s \in O(1)$, so that Theorem 1.1 states $C\text{-DAG}_1 \subseteq \text{P}^{\text{NP}[\log]}$. The intuition is as follows:

1. To apply admissible weighting functions and prove Theorem 4.1, a property we require[17] is that the length of any directed path in G be at most $D \in O(\log n)$. However, in the separator tree decomposition of Figure 3a, the longest path can in principle have $O(n)$ edges.
2. To address this, Figure 3b removes all "downward edges" with respect to the separator decomposition (i.e. edges $(v, w) \in E$ such that $\text{level}(v) < \text{level}(w)$ in the tree). Thus, the longest path now goes from a leaf to the root t, with each edge followed monotonically decreasing the current level[18]. Since our separators are *balanced*, any such path has length $O(\log n)$, as desired.
3. Of course, this breaks the logic of the C-DAG G itself. To correct this, we apply three ideas.
    (a) *Create node copies.* Each node of G (say, x_1) is split into multiple copies, each of which is hardcoded with a distinct possible output value of all its "ancestors" in the separator tree. In this example, x_1 is split into 64 copies of form $x_1^{z_1,z_2,z_3}$, over all $z_1, z_2, z_3 \in \{0,1\}^2$. Here, z_1 is intended to capture the outputs of u_1 and u_2, z_2 the outputs of v_1 and v_2, and z_3 the outputs of x_1 and x_2. Of these, x_1 only depends directly on the first bit of z_2 according to G;

[17] While necessary, this property itself is *not* sufficient; we use it here to ease the discussion. More accurately, we require that for any v, $|\text{Desc}(v)| \leq O(\log n)$. This latter property is trickier to attain, and does not follow from $D \in O(\log n)$.

[18] More accurately, the level is non-increasing with each edge followed. This is because the construction allows edges between pairs of vertices in the same supervertex (Equation (22)). This can incur an overhead in path length scaling with s, the separator size, which we ignore for this intuitive discussion.

all other bits in z_1, z_2, z_3 are irrelevant for x_1, and are included only to make the construction systematic. (They will be removed shortly when moving to G^* in Section 4.2.2.)

(b) *Add upward shortcuts.* Add new "upward" edges via Equation (22). In words, this roughly means that if u is an ancestor of v in G, but v occurs closer to the root than u, then we add upward shortcut edge (u, v) to E'. In our example, we connect $x_1^{z_1,z_2,z_3}$ to all (copies of) descendants of preimage($x_1^{z_1,z_2,z_3}$) which are *in the same unique superpath* from $x_1^{z_1,z_2,z_3}$ up to t (such as $v_2^{z_1,z_2}$). The careful reader may notice that $x_1^{z_1,z_2,z_3}$ does *not* have an edge to $x_2^{z_1,z_2,z_3}$. This is why x_2 has superscript z_3; this enumerates over all possible outputs it may have received from x_1.

(c) *Orchestrate the madness.* All copies of all nodes send their output to t via the dashed edges in Figure 3b. Roughly, t now selects, out of all the possible computation paths created via node copies, which is the "right" path. In our example, the "right" path can start with any copy of v_1 (e.g. $v_1^{00,00}$ or $v_1^{11,10}$, etc), since v_1 has in-degree 0 in G. Thus, all copies of v_1 encode the same NP query. Suppose this NP query outputs $b \in \{0, 1\}$. Then, the "right" path next utilizes any copy of x_1 of form $x_1^{z_1,bz_{2,2},z_3}$ (first bit of z_2 is b). And so forth. This "selection" of the "right path" is executed when t calls ComputeOutput($y_2 \mid \varepsilon$).

An explicit run-through. For concreteness, we now trace through t's call to ComputeOutput($y_2 \mid \varepsilon$) for Figure 3b:

```
ComputeOutput($y_2 \mid \varepsilon$):
    $z_{1,1} \leftarrow$ out-wire[$u_1^{00}$]
    $z_{1,2} \leftarrow$ out-wire[$u_2^{z_{1,1}0}$]
    return ComputeOutput($y_2 \mid z_1$):
        $z_{2,1} \leftarrow$ out-wire[$w_1^{z_1,00}$]
        $z_{2,2} \leftarrow$ out-wire[$w_2^{z_1,z_{2,1}0}$]
        return ComputeOutput($y_2 \mid z_1, z_2$):
            $z_{3,1} \leftarrow$ out-wire[$y_1^{z_1,z_2,00}$]
            $z_{3,2} \leftarrow$ out-wire[$y_2^{z_1,z_2,z_{3,1}0}$]
            return ComputeOutput($y_2 \mid z_1, z_2, z_3$):
                return $z_{3,2}$
```

Remark 4.10 (Promise gaps)**.** While the *size* of the verification circuit at any node $u_{S,i} \in V$ grows under the mapping to $v_{S,i}^{z_1,\ldots,z_d} \in V'$ (since the latter takes in more wires), the underlying verification procedures at each $u_{S,i}$ and $v_{S,i}^{z_1,\ldots,z_d}$ are identical, up to the latter's use of Rule 4.8 to decide on-the-fly which input wires to use based on $(z_1, \ldots, z_d)$. Thus, the *promise gaps* at each $u_{S,i}$ and $v_{S,i}^{z_1,\ldots,z_d} \in V'$ are also identical. When C is a promise class allowing error reduction, this is not of consequence; however, for StoqMA, which is not known to have error reduction, this observation allows us to keep the exponents in Theorem 1.4 at $O(s(n)\log^2 n)$ (versus $O(s^2(n)\log^2 n)$, since the promise gaps at each $v_{S,i}^{z_1,\ldots,z_d}$ node still scale as $1/\operatorname{poly}(n)$ due to this observation, not $1/\operatorname{poly}(|V''|)$.

Remark 4.11 (For StoqMA)**.** ComputeOutput($v \mid \varepsilon$) (and thus t) runs in DTIME(poly(V'')) to stitch together the answers of all other nodes of V. Thus, when $C =$ StoqMA, the action of t can also be viewed as a special case of a StoqMA computation (i.e. the StoqMA "verifier" for t would ignore its proof, use its classical gates to simulate ComputeOutput($v \mid \varepsilon$), and then output either $|+\rangle$ or $|-\rangle$ depending on whether it wishes to accept or reject, respectively.) Thus, in this case, all nodes of G'' are valid $C =$ StoqMA nodes, so G'' is a valid StoqMA-DAG.

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 3b, 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 3a, encodes the same query in Figure 3b, 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 $\mathrm{Anc}(u)$ denotes the ancestors of u in G, i.e. the set of queries u depends on, define

$$D_u := \mathrm{Anc}(u) \cap \bigcup_{j=1}^{d} S_j, \tag{23}$$

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 $h_v : D_u \to \{0,1\}$ 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 kth 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. preimage(v_1) = 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 $h_v : D_u \to \{0,1\}$), we will merge them. Formally, the merge is accomplished by Algorithm 3, which simultaneously computes an admissible weighting function. Henceforth, denote $(G^*, f^*) := \textsc{Merge}(G'')$.

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

```
function MERGE($G'' = (V'', E'')$)
    $G_1 \leftarrow G''$, $V_1 \leftarrow V''$, $E_1 \leftarrow E''$, $f_1 \leftarrow \omega$        ▷ for weighting function $\omega$ from Lemma 4.6
    $i \leftarrow 1$
    while $\exists v_1, v_2 \in V_i$ such that preimage$(v_1)$ = preimage$(v_2)$ and $h_{v_1} = h_{v_2}$ do
        Choose any such $v_1, v_2$ such that $v :=$ preimage$(w_1)$ is furthest from root in separator tree $T$.
        Create copy $v^*$ of $v$ with $h_{v^*} := h_{v_1} = h_{v_2}$.
        $V_{i+1} \leftarrow V_i \setminus \{v_1, v_2\} \cup \{v^*\}$
        Replace out-wire[$v_1$] and out-wire[$v_2$] in the logic of nodes in $V_{i+1}$ with out-wire[$v^*$].
        $E_{i+1} \leftarrow \{(r(x), r(y)) \mid (x,y) \in E_i\}$, where $r(x) := \begin{cases} v^*, & \text{if } x \in \{v_1, v_2\} \\ x, & \text{else} \end{cases}$
        Update $f_{i+1} : V_{i+1} \to \mathbb{R}$ such that $f_{i+1}(x) := \begin{cases} f_i(v_1) + f_i(v_2), & \text{if } x = v^* \\ f_i(x), & \text{else} \end{cases}$
    return $(G_i, f_i)$
```

Remark 4.12. 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 3a being mapped to $v_1^{**,**}$ in Figure 3c. Intuitively, this captures the fact that since v_1 is in-degree 0 in Figure 3a, the query at all copies $v_1^{z_1,z_2}$ of Figure 3b is identical, regardless of how z_1, z_2 are set.

Lemma 4.13. *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 4.6).*

Proof. We prove the lemma inductively on the iteration number, i. By Lemma 4.6, $f_1 = \omega$ is c-admissible, and trivially $W_{f_1}(G_1) = W_\omega(G'')$. In the induction step, Line 10 of Algorithm 3 straightforwardly yields $W_{f_i}(G_i) = W_{f_{i+1}}(G_{i+1})$. We show c-admissibility for f_{i+1}, i.e. that $f(v) \geq 1 + c\sum_{w\in\Gamma(v)} f(w)$ (Equation (10)) holds for all $v \in V_{i+1}$, where recall $\Gamma(v)$ is the set of children of v. Since the admissibility condition only depends on $\Gamma(v)$, it suffices to consider two cases: $v = v^*$ and v is a parent of v^*. First, if $v = v^*$, we have $\Gamma(v^*) = \Gamma(v_1) \cup \Gamma(v_2)$, and thus

$$f_{i+1}(v^*) = f_i(v_1) + f_i(v_2) \geq 2 + c \sum_{u\in\Gamma(v^*)} f_i(u) = 2 + c \sum_{u\in\Gamma(v^*)} f_{i+1}(u) \tag{24}$$

since f_i was c-admissible and since only v^* is altered in round i. Second, if v is a parent of v^*, then v is a parent of at least one of v_1 or v_2 by the construction of Algorithm 3. But by definition of the edge set of V'' (Equation (22)), v is a parent of v_1 if and only if v is a parent of v_2. Thus, v was a parent of both v_1 and v_2 in round i. The claim now follows since we set $f_{i+1}(v^*) = f_i(v_1) + f_i(v_2)$. □

4.2.3 Correctness

We now prove correctness, in the process establishing the Compression Lemma (Lemma 4.7). For this, we first require the following lemma, which shows how to efficiently map any given correct query string for G^* to a correct query string for G. Below, ComputeOutput 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 4.14. *Let $x^* : V^* \to \{0,1\}$ be a correct query string for G^*. Define* ComputeOutput* *to be* ComputeOutput*, except with each call to* out-wire *on Line 7 replaced by looking up the corresponding bit of x^*. Define string $x : V \to \{0,1\}$ such that bit $x(v) :=$* ComputeOutput*$(v \mid \varepsilon)$*. Then, x is a correct query string for G.*

Proof. Recall $|V| = n$, and that by Definition 3.3, 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 $(v_1, \ldots, v_n)$ on V fixed by Algorithm 1. We prove the claim inductively for $t \in (1, \ldots, n)$.

By the topological order, the base case $v_1 \in V$ has in-degree 0, i.e. takes no inputs. Thus, by Remark 4.12, 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. This bit will then be returned for v_1 once Line 4 executes, as desired.

For the inductive case, let $t \geq 2$ and assume $x_1, \ldots, x_{t-1}$ satisfy the induction hypothesis, i.e. they could[19] be produced by the first $t-1$ iterations of Evaluate. We now need to argue that a correct execution of Evaluate could set $x_t =$ ComputeOutput*$(v_t \mid \varepsilon)$. By design, ComputeOutput*$(v_t \mid \varepsilon)$ computes $z_1, \ldots, z_d$ and then returns[20] $x^*(v_t^{z_1,\ldots,z_d})$.

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. (Formally, $D_u := \mathrm{Anc}(u) \cap \bigcup_{j=1}^d S_j$ in Equation (23).) We claim that $z := z_1, \ldots, z_d$ 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 ComputeOutput*(,) and may also contain bits corresponding to vertices not in D_{v_t}.) To see this, fix any $u \in D_{v_t} = \mathrm{Anc}(v_t) \cap \bigcup_{j=1}^d S_j$, and let

[19] We say "could" because Evaluate is nondeterministic (due to potential invalid queries when C is a promise class).

[20] Technically, the algorithm actually returns $x^*(v_t^{z_1,\ldots,z_{d-1},z'_d})$, where $z'_d \in \{0,1\}^s$ is an "intermediate string" defined as follows: If node v_t was the kth node in the topological order for supervertex S, then the first $k-1$ bits of z'_d have been assigned by Line 7 of ComputeOutput*, 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 $v_t^{z_1,\ldots,z'_d}$ and $v_t^{z_1,\ldots,z_{d-1},z''_d}$ 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^*.

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, respectively[21]. Now, recall ComputeOutput*$(v_t \mid \varepsilon)$ recursively traverses the path $S_1, \ldots, S_d$, where $u \in S_i$ and $v_t \in S_d$ for $1 \leq i \leq d$. Thus, the operations performed by ComputeOutput*$(v_t \mid \varepsilon)$ and ComputeOutput*$(u \mid \varepsilon)$ are identical in the first i recursions. Once $m = i$ (i.e. conditioning strings $z_1, \ldots, z_i$ have been set), $z(u)$ is returned by ComputeOutput*$(u \mid \varepsilon)$ on Line 4, whereas ComputeOutput*$(v_t \mid \varepsilon)$ returns $z(v_t)$ if $i = d$ and continues recursively otherwise.

We thus conclude ComputeOutput*$(v_t \mid \varepsilon)$ returns $x^*(v_t^{z_1,\ldots,z_d})$ with $z(u) = x(u)$ for all $u \in D_{v_t}$. Recall now by Rule 4.8 that, in order for $v_t^{z_1,\ldots,z_d} \in V^*$ to simulate the query at $v_t \in V$, it computes the input on any wire u into v_t via ComputeOutput*$(u \mid z_1, \ldots, z_d)$. 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 3a, 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 ComputeOutput*$(v_t \mid \varepsilon)$ and ComputeOutput*$(u \mid \varepsilon)$ perform exactly the same operations in the first d recursions. Therefore, both executions compute the same values $z_1, \ldots, z_d$. Subsequently, ComputeOutput*$(u \mid \varepsilon) = x(u)$ calls ComputeOutput*$(u \mid z_1, \ldots, z_d)$ on Line 8 during the dth recursion and returns its value. In other words, $x(u) =$ ComputeOutput*$(u \mid z_1, \ldots, z_d)$. But by Rule 4.8, in order to simulate its input on the incoming wire corresponding to u, $v_t^{z_1,\ldots,z_d}$ uses ComputeOutput*$(u \mid z_1, \ldots, z_d) = x(u)$. Then, since $u \in \text{Anc}(v_t)$, 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 $v_t^{z_1,\ldots,z_d}$ must be set correctly, and thus $x(v_t)$ is also correct. □

We finally restate and prove the main lemma of this section.

Lemma 4.7. *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* Evaluate$(G) =$ Evaluate(G^*) *(irrespective of nondeterministic choices in Algorithm 1). As required by the definition of C-DAG (Definition 3.1), each node of G^* corresponds to a verification circuit of size* poly$(|V^*|)$.

Proof of Lemma 4.7. 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 $n = |V|$), since there are at $2^{O(sD)}$ 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 4.6, we have $W_{f^*}(G^*) = W_\omega(G'') \leq (c+1)^{O(sD)}n$, where the equality is from Lemma 4.13, and the inequality since every node in G'' has at most $O(sD)$ descendants. Correctness follows from Lemma 4.14 and the fact that the output of G^*, by definition of node t, is ComputeOutput$(v \mid \varepsilon)$. Finally, each verification circuit corresponding to a node in V^* has size poly(V^*); the largest such verification circuit corresponds to node t, which takes in wires from *all* other vertices in G^*, and calls ComputeOutput$(v \mid \varepsilon)$ (which takes time poly$(|V^*|)$). □

[21] 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 ComputeOutput* 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.

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 [Got95]; 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 $G = (V = \{v_1, \ldots, v_N\}, E)$ be a C-DAG-instance with c-admissible weighting function f. Recall by Definition 3.1 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 $t : \{0,1\}^N \times (\mathcal{B}^{\otimes m})^{\times N} \mapsto \mathbb{R}$ such that

$$t(x, \psi_1, \ldots, \psi_N) := \sum_{i=1}^{N} f(v_i)\underbrace{\big(x_i \Pr[Q_i(z_i(x), \psi_i) = 1] + (1 - x_i)\gamma\big)}_{g(x_i, z_i(x), \psi_i)}, \tag{25}$$

where $\gamma := (\alpha + \beta)/2$, and where $z_i(x)$ 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

$$T := \max_{\substack{x \in \{0,1\}^N \\ |\psi_1\rangle, \ldots, |\psi_N\rangle \in \mathcal{B}^{\otimes m}}} t(x, \psi_1, \ldots, \psi_N). \tag{26}$$

In Section 4.3.2 we show how to approximate T using $O(\log W_f(G))$ C-queries and in Section 4.3.3 we prove that if $t(x, \psi_1, \ldots, \psi_N) \approx T$, then x is a correct query string.

4.3.2 Step 2: Approximating T

In order to apply binary search to approximate T (see proof of Theorem 4.1), 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

$$\begin{aligned} \Pi_{\text{yes}} &= \{(t, s) \mid t : \{0,1\}^N \times \left(\mathcal{B}^{\otimes m}\right)^{\times N} \mapsto \mathbb{R} \text{ and } T \geq s\} & (27) \\ \Pi_{\text{no}} &= \{(t, s) \mid t : \{0,1\}^N \times \left(\mathcal{B}^{\otimes m}\right)^{\times N} \mapsto \mathbb{R} \text{ and } T \leq s - \varepsilon\}, & (28) \end{aligned}$$

for T as in Equation (26), and $\varepsilon : \mathbb{Z} \mapsto \mathbb{R}^{\geq 0}$ a fixed function of N (i.e. by ε we mean $\varepsilon(N)$).

Lemma 4.15. *Let $C \in \mathcal{QV}^+$. Define $W := \sum_{i=1}^N f(v_i)$ for weighting function f from Equation* (25)*, 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 $t(x, \psi_1, \ldots, \psi_N)$ directly (note that $t : \{0,1\}^N \times (\{0,1\}^{\otimes m})^{\times N} \mapsto \mathbb{R}$ in this case).

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

$$t'(x,\psi_1,\ldots,\psi_N) := \frac{1}{W}t(x,\psi_1,\ldots,\psi_N) = \sum_{i=1}^{N} p_i \cdot g(x_i,z_i,\psi_i), \tag{29}$$

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

$$\max_{\text{proofs } |\psi\rangle} \Pr[V \text{ outputs } 1 \mid |\psi\rangle] = T'. \tag{30}$$

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 $\varepsilon/W \geq 1/\operatorname{poly}(N)$ since $W \leq \operatorname{poly}(N)$ by assumption.[22]

V has proof space $\mathcal{X} \otimes \mathcal{Y}_1 \otimes \cdots \otimes \mathcal{Y}_N$ with $\mathcal{X} = \mathcal{B}^{\otimes N}$ and $\mathcal{Y}_i = \mathcal{B}^{\otimes m}$. A subtle point here is that function t takes as part of its input a sequence $(|\psi_1\rangle,\ldots,|\psi_N\rangle)$, whereas in Equation (30), V takes in a joint (potentially entangled) proof $|\psi\rangle$ across proof registers $\mathcal{Y}_1 \otimes \cdots \otimes \mathcal{Y}_N$. However, due to the construction of V below, we shall see that without loss of generality, Equation (30) is attained for tensor product states $|\psi\rangle = |\psi_1\rangle \otimes \cdots \otimes |\psi_N\rangle$, which is equivalent to sequence $(|\psi_1\rangle,\ldots,|\psi_N\rangle)$, as desired.

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

```
1: Measure $\mathcal{X}$ in standard basis to obtain string $x$.
2: Select random $i$ according to distribution $p_i$.[23]
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 $\gamma$.
```

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 $|\psi\rangle = |x\rangle_{\mathcal{X}}|\psi'\rangle_{\mathcal{Y}_{1,\ldots,N}}$. Then, since we only measure a single local verifier Q_i (at random, Step 4), we have

$$\Pr[V \text{ outputs } 1 \mid |\psi\rangle] = t'(x,\sigma_1,\ldots,\sigma_N) \quad \text{where} \quad \sigma_i := \operatorname{Tr}_{\bigotimes_{j\neq i}\mathcal{Y}_j}(|\psi'\rangle\langle\psi'|). \tag{31}$$

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 $|\psi\rangle = |\psi_1\rangle \otimes \cdots \otimes |\psi_N\rangle$, and so Equation (30) holds. This completes the proof.

Two final remarks are needed for specific choices of C: (1) For $C = QMA(2)$, separable σ_i are obtained by interpreting the first proof register as $\mathcal{X} \otimes \mathcal{Y}_1^1 \otimes \cdots \otimes \mathcal{Y}_N^1$ and the second as $\mathcal{Y}_1^2 \otimes \cdots \otimes \mathcal{Y}_N^2$ (so that the joint proof is unentangled across this cut by assumption), where proof $\mathcal{Y}_i$ has the registers Y_i^1 and Y_i^2. (2) For $C = \text{StoqMA}$, V is indeed a stoquastic[24] verifier since:

- Via its $|+\rangle$ ancillae states and ability to simulate measurement in the standard basis via the principle of deferred measurement, a stoquastic verifier can execute Steps 1 and 2 in the description of V.
- Each Q_i is by definition a stoquastic verifier (see Remark 4.11), and Step 4 of the V simply returns the output of some Q_i *without* postprocessing, i.e. the output qubit of Q_i is simply swapped into the output qubit of V.
- By definition of StoqMA, $1/2 \leq \gamma \leq 1$ with γ requiring (without loss of generality) at most

[22] For $C = \text{QMA}(2)$, we implicitly use the nontrivial error reduction of QMA(2) due to Harrow and Montanaro [HM13].

[23] This requires being able to sample from the distribution p_i, which in general cannot be done efficiently. However, we can approximate $p_i \approx k_i/2^{\operatorname{poly}(N)}$, allowing efficient sampling without changing the distribution significantly.

[24] Briefly, a stoquastic verifier [BBT06b] takes in a poly-size quantum proof and poly many ancillae set to $|0\rangle$ and $|+\rangle$ states, runs poly many classical gates (i.e. Pauli X, CNOT, and Toffoli gates), and finally applies a single Hadamard gate to its output qubit just before measuring it in the standard basis.

logarithmic bits of precision to specify. Thus, Step 6 of V can be simulated by a stoquastic verifier which, with appropriate conditioning, swaps into its output qubit either an ancillae qubit set to $|0\rangle$ (accepted with probability 1/2, due to the final H gate on the measurement qubit of a stoquastic verifier) or an ancillae qubit set to $|+\rangle$ (accepted with probability 1).

□

Note that Lemma 4.15 says nothing about *why* we want to approximate T, i.e. what the optimal argument x buys us. This is the purpose of Section 4.3.3.

4.3.3 Step 3: Correct Query String

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

Lemma 4.16. *Define $\eta := (\alpha - \beta)/2$, and let f be η^{-1}-admissible. If $t(x, \psi_1, \ldots, \psi_N) > T - \eta$, then x is a correct query string.*

Proof. Assume there exists a $v_i \in V$ such that x_i is incorrect. We show that there exist x', $|\psi_1'\rangle, \ldots, |\psi_N'\rangle \in \mathcal{B}^{\otimes N}$ such that $t(x', \psi_1', \ldots, \psi_N') \geq t(x, \psi_1, \ldots, \psi_N) + \eta$, obtaining a contradiction. A subtle but useful fact we exploit is that t takes in a *sequence* $(\psi_1, \ldots, \psi_N)$; this allows us to locally update each ψ_i to some ψ_i' as follows. Define x',$|\psi_1'\rangle$,…,$|\psi_N'\rangle$ such that $x_i' = \overline{x_i}$ (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 $\Pr[Q_i(z_i(x'), \psi_i') = 1]$.

Now, if $x_i = 0$, then

$$g(x_i, z_i(x), \psi_i) = x_i \Pr[Q_i(z_i(x), \psi_i) = 1] + (1 - x_i)\gamma = \gamma. \tag{32}$$

Since we assumed x_i was incorrect, $z_i(x) \in \Pi^i_{\text{yes}}$. Thus, there exists a $|\psi_i'\rangle$ such that

$$g(x_i', z_i(x'), \psi_i') = x_i' \Pr[Q_i(z_i(x'), \psi_i') = 1] + (1 - x_i')\gamma = \Pr[Q_i(z_i(x'), \psi_i') = 1] \geq \alpha. \tag{33}$$

Conversely, if $x_i = 1$, we have $z_i(x) \in \Pi^i_{\text{no}}$, and thus

$$g(x_i, z_i(x), \psi_i) = x_i \Pr[Q_i(z_i(x), \psi_i) = 1] + (1 - x_i)\gamma \leq \beta, \tag{34}$$

whereas for any $|\psi_i'\rangle$,

$$g(x_i', z_i(x'), \psi_i') = x_i' \Pr[Q_i(z_i(x'), \psi_i) = 1] + (1 - x_i')\gamma = \gamma. \tag{35}$$

Thus, flipping x_i to $\overline{x_i}$ increases the ith term in the sum comprising t by at least $\eta \cdot f(v_i)$ (recall

$\gamma = (\alpha + \beta)/2)$. Therefore,

$$
\begin{aligned}
&t(x', \psi'_1, \dots, \psi'_N) - t(x, \psi_1, \dots, \psi_N) && (36)\\
&= \sum_{j=1}^{N} f(v_j) g(x'_j, z_j(x'), \psi'_j) - \sum_{j=1}^{N} f(v_j) g(x_j, z_j(x), \psi_j) && (37)\\
&= f(v_i)\big(\underbrace{g(x'_i, z_i(x), \psi'_i) - g(x_i, z_i(x), \psi_i)}_{\geq \eta}\big) + \sum_{(v_i,v_j)\in E} f(v_j)\big(\underbrace{g(x'_j, z_j(x'), \psi_j) - g(x_j, z_j(x), \psi_j)}_{\geq -1}\big) && (38)\\
&\geq \eta \cdot f(v_i) - \sum_{(v_i,v_j)\in E} f(v_j) && (39)\\
&= \eta\Bigg(f(v_i) - \eta^{-1} \sum_{(v_i,v_j)\in E} f(v_j)\Bigg) && (40)\\
&\geq \eta, && (41)
\end{aligned}
$$

where the second statement holds since $g(\cdot) \in [0, 1]$ and since flipping x_i to x'_i only affects the immediate children of v_i (since each z_j function only depends on the direct inputs to node v_j), and the last statement since 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 4.1.

Proof of Theorem 4.1. First, apply the Compression Lemma (Lemma 4.7) 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 (25), whose maximum value we denoted T (Equation (26)). By Lemma 4.16, 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 $C \in \{\mathrm{NP}, \mathrm{MA}, \mathrm{QCMA}, \mathrm{QMA}, \mathrm{QMA}(2)\}$, where for QMA(2) we use [HM13]) that $\alpha = 2/3$ and $\beta = 1/3$, so that $\eta^{-1} = 6$. By Lemma 4.13, f^* is c-admissible for any $c \geq 2$, and hence η^{-1}-admissible. Third, use Lemma 4.15 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 $\widetilde{T}$ denote this estimate of T. Fourth, make a final C-query via Lemma 4.15 to decide whether there exists a correct query string x and proofs $|\psi_1\rangle, \dots, |\psi_N\rangle$, such that $t(x, \psi_1, \dots, \psi_N) \geq \widetilde{T}$ and for which $x_N = 1$, and return its answer. (Recall that x_N, by definition, encodes the output of the C-DAG.) □

4.4 The case of StoqMA

We are only able to show a weaker version of Theorem 4.1 for StoqMA, due to the fact that error reduction for StoqMA is not known (i.e. one cannot assume completeness/soundness 2/3 and 1/3). Specifically, Lemmas 4.7, 4.15 and 4.16 still hold for $C = \mathrm{StoqMA}$. However, as amplification of StoqMA's promise gap is not known (see, e.g. [AGL20]), we cannot assume $\eta = \Omega(1)$ in the proof of Theorem 4.1. If we instead use $\eta = 1/\mathrm{poly}(n)$ (note the use of n versus N here; see Remark 4.10), Lemmas 4.15 and 4.16 require a $\mathrm{poly}(n)$-admissible weighting function. However, for any c-admissible weighting function f,

$W_f(G) \geq c^{\mathrm{depth}(G)}$, which is superpolynomial when $\mathrm{depth}(G) = \omega(1)$ and $c = \mathrm{poly}(n)$. Thus, for StoqMA we can only prove the following weaker analogue of Theorem 1.1:

Theorem 1.4. *Fix $C = \mathrm{StoqMA}$ and any efficiently computable function $s : \mathbb{N} \to \mathbb{N}$. Then,*

$$C\text{-DAG}_s \in \mathrm{DTIME}\left(2^{O(s(n)\log^2 n)}\right)^{C[s(n)\log^2 n]}. \tag{4}$$

Proof. This follows analogously to Theorem 1.1, but with $c \in \mathrm{poly}(n)$ (see Remark 4.10) instead of $c \in O(1)$, which incurs an additional log factor in the exponent. □

Akin to Corollary 1.3, it follows that:

Corollary 4.17. *For $C = \mathrm{StoqMA}$, $C\text{-DAG}_{\log^k} \in \mathrm{QP}^{C[\log^{k+2}(n)]}$ for all constants $k \in \mathbb{N}$.*

4.5 Query Graphs of Bounded Depth

One can ask whether there are other kinds of graphs for which we can apply the techniques developed in this section. Using the weighting function ρ from Lemma 4.6, we obtain the following results for query graphs of bounded depth.

Definition 4.18 (C-DAG$_d$)**.** Let $d : \mathbb{N} \to \mathbb{N}$ be an efficiently computable function. Then, C-DAG$_d$ is defined as C-DAG, except that G has depth scaling as $O(d(n))$, for n the number of nodes used to specify the C-DAG instance.

We caution that this notation is very close to that of C-DAG$_s$ — the d in C-DAG$_d$ distinguishes that here we are considering bounded depth (as opposed to bounded separator number with C-DAG$_s$). The union of C-DAG$_d$ over all polynomials $d : \mathbb{N} \mapsto \mathbb{N}$ equals C-DAG.

The next theorem was shown by Gottlob [Got95] for $C = \mathrm{NP}$ and $d(n) = \log^i n, i \in \mathbb{N}$. We strengthen it for NP and simultaneously extend it to the quantum setting.

Theorem 1.5. *Let $d : \mathbb{N} \to \mathbb{N}$ be an efficiently computable function. For $C \in \{\mathrm{NP}, \mathrm{NEXP}, \mathrm{QMA}_{\mathrm{exp}}\}$, $C\text{-DAG}_d \subseteq \mathrm{P}^{C[d(n)\log(n)]}$, and for $C \in \mathcal{QV}^+$,*

$$C\text{-DAG}_d \subseteq \mathrm{DTIME}\left(2^{O(d(n)\log(n))}\right)^{C[d(n)\log(n)]}.$$

Proof. Follows analogously to Theorem 1.1, except we do not need to apply the graph transformation. It suffices to use Lemmas 4.15 and 4.16 directly with the weighting function ρ from Lemma 4.6. For $C \in \{\mathrm{NP}, \mathrm{NEXP}, \mathrm{QMA}_{\mathrm{exp}}\}$, we do not need to increase the runtime of the base class to the total weight $W_\rho(G)$ because the queries for Lemma 4.15 can be performed exactly or with exponential precision in the case of $\mathrm{QMA}_{\mathrm{exp}}$. □

Corollary 4.19. *For $C \in \mathcal{QV}^+$ and $d \in O(1)$, C-DAG$_d$ is $\mathrm{P}^{C[\log]}$-complete.*

Proof. Containment in $\mathrm{P}^{C[\log]}$ is given by Theorem 1.5. That C-DAG$_d$ for $d \in O(1)$ is $\mathrm{P}^{C[\log]}$-hard follows analogously to Theorem 1.2 (in particular, since $\mathrm{P}^{C[\log]} \subseteq \mathrm{P}^{\|C}$ for general C [Bei91], and since parallel queries correspond to a constant depth query graph). □

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 [WBG20], thus directly yielding hardness results for the APX-SIM problem (Definition 2.6). We first give required definitions in Section 5.1. Section 5.2 and Section 5.3 state and prove the main result of this section, the Generalized Lifting Lemma (Lemma 5.3). Finally, Section 5.4 shows how to apply the Generalized Lifting Lemma to obtain hardness results for APX-SIM (see Definition 2.6).

5.1 Definitions

The following definitions were introduced in [WBG20] to allow one to abstractly speak about large classes of circuit-to-Hamiltonian mappings. This allows the Lifting Lemma of [WBG20], as well as its generalized version shown in Section 5.2 (Lemma 5.3), 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.

Definition 5.1 (Conformity [WBG20])**.** 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.

For example, if H is a 2-local Hamiltonian on a 2D square lattice, then P conforms to H if $H + P$ is also a 2-local Hamiltonian on a 2D square lattice. Next, define $\mathrm{U}(\mathcal{X})$ as the set of unitary operators acting on space $\mathcal{X}$.

Definition 5.2 (Local Circuit-to-Hamiltonian Mapping [WBG20])**.** Let $\mathcal{X} = (\mathbb{C}^2)^{\otimes p}$ and $\mathcal{Y} = (\mathbb{C}^2)^{\otimes q}$. A map $H_{\mathrm{w}} : \mathrm{U}(\mathcal{X}) \mapsto \mathrm{Herm}(\mathcal{Y})$ 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) $H_{\mathrm{w}}(U) \succeq 0$ has a non-trivial null space, i.e. $\mathrm{Null}(H_{\mathrm{w}}(U)) \neq 0$. 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 $S_{\mathrm{pre}} \subseteq \mathcal{X}$ and $S_{\mathrm{post}} \subseteq \mathcal{Y}$ denote the sets of input states to U satisfying the structure enforced by $H_{\mathrm{w}}(U)$ (e.g. ancillae initialized to zeroes), and null states of $H_{\mathrm{w}}(U)$, respectively. Then, there exist projectors M_1 and P_L, projector M_2 conforming to $H_{\mathrm{w}}(U)$, and a bijection $f : S_{\mathrm{pre}} \mapsto S_{\mathrm{post}}$, such that for all $i \in \{1, 2\}$ and $|\phi\rangle \in S_{\mathrm{pre}}$, the state $|\psi\rangle = f(|\phi\rangle)$ satisfies

$$\mathrm{Tr}\left(|0\rangle\langle 0|_i (U_L U_{L-1} \dots U_1)|\phi\rangle\langle\phi|(U_L U_{L-1} \dots U_1)^\dagger\right) = \mathrm{Tr}\left(|\psi_L\rangle\langle\psi_L| M_i\right), \tag{42}$$

   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 $g : \mathbb{N} \times \mathbb{N} \mapsto \mathbb{R}$ such that

$$\| P_L|\psi\rangle \|_2^2 = g(p, L) \text{ for all } |\psi\rangle \in \mathrm{Null}(H_{\mathrm{w}}(U)), \tag{43}$$

$$M_i = P_L M_i P_L. \tag{44}$$

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

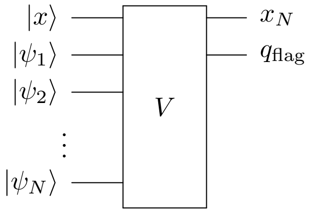


Figure 4: A depiction of the circuit V constructed in Lemma 4.15, with two minor modifications for our purposes here. First, the second wire above denotes the output wire of V, and is relabbeled q_{flag} here. Second, we assume without loss of generality that V outputs the Nth bit of $x \in \{0,1\}^N$ on the first wire above, labelled x_N. For simplicity, we depict the proofs $|\psi_i\rangle$ above in tensor product, but we make no such *a priori* assumption in any of our proofs.

To gain intuition about Definition 5.2, consider the simplest case of Kitaev's 5-local construction applied to a QMA verification circuit $U = U_L \cdots U_1$ [KSV02]. Then[25], $H_{\text{w}}(U) = H_{\text{in}} + H_{\text{prop}} + H_{\text{stab}}$, since recall the null space of $H_{\text{w}}(U)$ is precisely the set of all correctly initialized history states. (Notably, the term H_{out} is omitted.) The sets S_{pre} and S_{post} correspond to the correctly initialized inputs to U (i.e. of form $|\psi\rangle_A|0\cdots0\rangle_B$ for some proof $|\psi\rangle_A$ and the ancilla register B set to all zeroes) and all correctly initialized history states[26] $|\psi_{\text{hist}}\rangle$, respectively. The projector $P_L = |L\rangle\langle L|_C$ projects onto timestep L in clock register C, with $g(p,L) = 1/(L+1)$. Finally, $M_i = |0\rangle\langle 0|_{A_i}$ (for A_i the ith qubit of register A, and where the projection onto time step L has already happened due to the use of $|\psi_L\rangle$ in Equation (42)).

5.2 The Generalized Lifting Lemma

Lemma 5.3 (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 4, be the verification circuit constructed in Lemma 4.15, given (G^*, f^*). Define shorthand Δ for $\Delta(H_{\text{w}}(V))$. Fix a local circuit-to-Hamiltonian mapping H_{w}, and assume the notation in Definition 5.2. Fix any function $\alpha : \mathbb{N} \mapsto \mathbb{N}$ such that*

$$\alpha > \max\left(\frac{4\|M_2\|}{\Delta}, \frac{\Delta}{3\|M_2\|^2}, 1\right). \tag{45}$$

Then, the Hamiltonian $H := \alpha H_{\text{w}}(V) + M_2$ satisfies:

- *If G is a YES instance, then for all $|\psi\rangle$ with $\langle\psi|H|\psi\rangle \leq \lambda_{\min}(H) + \frac{1}{\alpha^2}$,*

$$\langle\psi|M_1|\psi\rangle \leq \frac{1}{\alpha}\left[\frac{W}{\eta}\left(\frac{1}{\alpha} + \frac{12\|M_2\|^2}{\Delta}\right) + \frac{12\|M_2\|^2}{\Delta}\right]. \tag{46}$$

- *If G is a NO instance, then for all $|\psi\rangle$ with $\langle\psi|H|\psi\rangle \leq \lambda_{\min}(H) + \frac{1}{\alpha^2}$,*

$$\langle\psi|M_1|\psi\rangle \geq g(p,L) - \frac{1}{\alpha}\left[\frac{W}{\eta}\left(\frac{1}{\alpha} + \frac{12\|M_2\|^2}{\Delta}\right) - \frac{12\|M_2\|^2}{\Delta}\right], \tag{47}$$

for W and η defined in Lemma 4.15 and Lemma 4.16, respectively, and $g(p,L)$ defined in Definition 5.2.

[25] In [KSV02], H_{in} checks that all ancillae are set to $|0\rangle$ before the verification, H_{prop} checks that each step i in the verification follows from step $i-1$, H_{stab} ensures the clock register is correctly encoded, and H_{out} checks that the verifier accepts the given proof.

[26] Kitaev's history state [KSV02] encodes the history of the verification in *superposition*, i.e. as $|\psi_{\text{hist}}\rangle \propto \sum_{t=0}^{L} U_t \cdots U_1|\psi\rangle_A|0\cdots0\rangle_B|t\rangle_C$, where C is a clock register. This in contrast to the Cook-Levin theorem [Coo71; Lev73b], which encodes the history in a tableau.

Proof. The claim follows immediately by defining $\delta := 1/\alpha^2$, and then combining Lemma 5.4, Lemma 5.5, and Lemma 5.6 (all given subsequently in Section 5.3). Roughly, Lemma 5.4 first shows that any low-energy state of H must be "close" to a history state (formally, a null state of $H_{\mathrm{w}}(V)$, as per Definition 5.2). Lemma 5.5 shows that, in turn, any low-energy *history* state of H must have most of its weight on correct query strings. Finally, Lemma 5.6 combines the previous two lemmas, along with Definition 5.2, to obtain the claim, i.e. the ground state of H must encode the full computation represented by G, and thus a local measurement suffices to decide G. □

5.3 Lemmas required for proof of Lifting Lemma

We now give the three lemmas required for the proof of Lemma 5.3, all of which assume the notation for the latter. The first of these can be stated and proven identically to Lemma 22 of [WBG20], since it does not leverage any properties of V itself, but only the abstract definition of $H_{\mathrm{w}}(U)$. While the proof is simple, it uses the Extended Projection Lemma [KKR06; GY19]; for brevity we omit both here.

Lemma 5.4 ([WBG20])**.** *Fix any function $\alpha : \mathbb{N} \mapsto \mathbb{N}$ such that*

$$\alpha > \max\left(\frac{4\|M_2\|}{\Delta}, \frac{\Delta}{3\|M_2\|^2}, 1\right), \tag{48}$$

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 $|\phi\rangle \in \mathrm{Null}(H_{\mathrm{w}}(V))$ such that

$$\||\psi\rangle\langle\psi| - |\phi\rangle\langle\phi|\|_{\mathrm{tr}} \leq \frac{12\|M_2\|}{\alpha\Delta} \tag{49}$$

and where $|\phi\rangle$ has energy

$$\langle\phi|H|\phi\rangle \leq \lambda_{\min}(H) + \delta + \frac{12\|M_2\|^2}{\alpha\Delta}. \tag{50}$$

For the second lemma, Lemma 5.5, 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 5.5 is our version of Lemma 23 of [WBG20]; 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 4.15. (In particular, Lemma 23 of [WBG20] requires a somewhat involved argument using conditional probabilities to obtain soundness against entanglement across proofs.)

Lemma 5.5. *Suppose history state $|\phi\rangle \in \mathrm{Null}(H_{\mathrm{w}}(V))$ has preimage $|\psi_{\mathrm{in}}\rangle = f^{-1}(|\phi\rangle)$ (for bijection f from Definition 5.2), where $|\psi_{\mathrm{in}}\rangle$ has proof $|w_{\mathcal{X}\mathcal{Y}}\rangle$ with total amplitude p_{bad} on incorrect query strings in $\mathcal{X}$. Then,*

$$\langle\phi|H|\phi\rangle > \lambda_{\min}(H) + g(p, L)\frac{p_{\mathrm{bad}} \cdot \eta}{W}. \tag{51}$$

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

$$|w_{\mathcal{X}\mathcal{Y}}\rangle = \sum_{x\in X_-} \alpha_x|x\rangle_{\mathcal{X}}|\psi_x\rangle_{\mathcal{Y}} + \sum_{x\in X_+} \alpha_x|x\rangle_{\mathcal{X}}|\psi_x\rangle_{\mathcal{Y}}, \tag{52}$$

for $\sum_{x\in X_+\cup X_-}|\alpha_x|^2 = 1$, arbitrary unit vectors $\{|\psi_x\rangle\}_x$, and $p_{\mathrm{bad}} := \sum_{x\in X_-}|\alpha_x|^2$. Recall from Definition 5.2 that M_2 simulates the projector $|0\rangle\langle 0|_{q_{\mathrm{flag}}}$ via

$$\mathrm{Tr}\left(|0\rangle\langle 0|_2(U_L U_{T-1}\dots U_1)|\psi_{\mathrm{in}}\rangle\langle\psi_{\mathrm{in}}|(U_L U_{T-1}\dots U_1)^\dagger\right) = \mathrm{Tr}\left(|\phi_L\rangle\langle\phi_L|M_2\right), \tag{53}$$

(since we assumed in Figure 4 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

$$\begin{aligned}
\langle\phi|H|\phi\rangle &= \langle\phi|M_2|\phi\rangle && (54)\\
&= g(p,L)\langle\phi_L|M_2|\phi_L\rangle && (55)\\
&= g(p,L)\,\mathrm{Tr}\left(|\psi_{\mathrm{out}}\rangle\langle\psi_{\mathrm{out}}|\cdot|0\rangle\langle 0|_{q_{\mathrm{flag}}}\right) && (56)\\
&= g(p,L)\Pr[V \text{ rejects} \mid |w_{\mathcal{X}\mathcal{Y}}\rangle], && (57)
\end{aligned}$$

where the second statement follows from Equation (43) and Equation (44), and the third from Equation (53). By Equation (26) and Equation (29), 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 $|\psi'_{\mathrm{in}}\rangle$ be an input state containing this optimal proof $|w'_{\mathcal{X}\mathcal{Y}}\rangle$. Lemma 4.16 now yields[27]

$$\begin{aligned}
\langle\phi|H|\phi\rangle &> g(p,L)\left(\Pr[V \text{ rejects} \mid |w'_{\mathcal{X}\mathcal{Y}}\rangle] + \left(\sum_{x\in X_-}|\alpha_x|^2\right)\frac{\eta}{W}\right) && (58)\\
&= \langle\phi'|M_2|\phi'\rangle + g(p,L)\frac{p_{\mathrm{bad}}\cdot\eta}{W} && (59)\\
&\geq \lambda_{\min}(H) + g(p,L)\frac{p_{\mathrm{bad}}\cdot\eta}{W}, && (60)
\end{aligned}$$

where the first inequality (58) uses the fact that

$$\Pr[V \text{ accepts} \mid |w_{\mathcal{X}\mathcal{Y}}\rangle] \leq p_{\mathrm{good}}\cdot\frac{T}{W} + p_{\mathrm{bad}}\left(\frac{T}{W}-\frac{\eta}{W}\right) = \frac{T}{W} - \frac{p_{\mathrm{bad}}\cdot\eta}{W}. \qquad (61)$$

The second statement uses Equations (43) and (44), with $|\phi'\rangle := f(|\psi'_{\mathrm{in}}\rangle)$, and the last statement (60) uses $|\phi'\rangle \in \mathrm{Null}(H_{\mathrm{w}}(V))$ by the definition of f in Definition 5.2.

As a final aside, the proof above is written with the context of *quantum* verification classes such as $C = \mathrm{QMA}$ in mind. However, the same proof can be applied directly to (say) $C = \mathrm{NP}$ by embedding an NP verifier in the usual manner into a QMA verifier (i.e. the QMA verifier begins by measuring its proof in the standard basis via the principle of deferred measurement). Of course, even when $C = \mathrm{NP}$, the construction of this section still yields a genuinely quantum Hamiltonian H (as opposed to a Hamiltonian H diagonal in the standard basis), due to our use of circuit-to-Hamiltonian mappings H_{w}. □

Finally, the third lemma, Lemma 5.6, is our analog of Lemma 25 of [WBG20]. We follow the same high-level approach as the latter, but again, our proof here is simplified. This is because Lemma 4.16 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 4. (In contrast, [WBG20] needed the Commutative Quantum Union Bound to argue that all proofs are simultaneously correct.)

Lemma 5.6. *Consider any $|\psi\rangle$ satisfying $\langle\psi|H|\psi\rangle \leq \lambda_{\min}(H)+\delta$. If $\delta \leq 1/\alpha^2$, then*

- *if G^* is a YES instance, then*

$$\langle\psi|M_1|\psi\rangle \leq \frac{W}{\eta}\left(\delta + \frac{12\|M_2\|^2}{\alpha\Delta}\right) + \frac{12\|M_2\|^2}{\alpha\Delta} \qquad (62)$$

[27] We are implicitly using the fact that, as observed in the proof of Lemma 4.15, 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 $\mathcal{Y}$.

- *if G^* is a NO instance, then*

$$\langle\psi|M_1|\psi\rangle \geq g(p,L) - \frac{W}{\eta}\left(\delta + \frac{12\|M_2\|^2}{\alpha\Delta}\right) - \frac{12\|M_2\|^2}{\alpha\Delta} \tag{63}$$

Proof. We first use Lemma 5.4 to map, assuming $\delta \leq 1/\alpha^2$, $|\psi\rangle$ to a history state $|\phi\rangle \in \mathrm{Null}(H_{\mathrm{w}}(V))$ such that

$$\||\psi\rangle\langle\psi| - |\phi\rangle\langle\phi|\|_{\mathrm{tr}} \leq \frac{12\|M_2\|}{\alpha\Delta} \quad \text{and} \quad \langle\phi|H|\phi\rangle \leq \lambda_{\min}(H) + \delta + \frac{12\|M_2\|^2}{\alpha\Delta}. \tag{64}$$

We next use Lemma 5.5 to obtain that preimage $|\phi_{\mathrm{in}}\rangle = f^{-1}(|\phi\rangle)$ contains proof $|w_{1\cdots m}\rangle$ (see Equation (52)) with

$$p_{\mathrm{bad}} < \frac{W}{g(p,L)\cdot\eta}\left(\delta + \frac{12\|M_2\|^2}{\alpha\Delta}\right), \tag{65}$$

i.e. the total amplitude p_{bad} of $|w_{\mathcal{X}\mathcal{Y}}\rangle$ on incorrect query strings in $\mathcal{X}$ is bounded. But by Definition 3.1, if the query string $x_1\cdots x_N$ in $\mathcal{X}$ is correct, then x_N encodes the correct output of C-DAG G^*. Moreover, by design, V in Figure 4 always outputs x_N on its first wire. Thus,

$$\begin{aligned} \text{if } G^* \text{ is a YES instance} \quad &\Rightarrow \quad \mathrm{Tr}\left(|0\rangle\langle 0|_1 V|\phi_{\mathrm{in}}\rangle\langle\phi_{\mathrm{in}}|V^\dagger\right) \leq p_{\mathrm{bad}},\\ \text{if } G^* \text{ is a NO instance} \quad &\Rightarrow \quad \mathrm{Tr}\left(|0\rangle\langle 0|_1 V|\phi_{\mathrm{in}}\rangle\langle\phi_{\mathrm{in}}|V^\dagger\right) \geq 1 - p_{\mathrm{bad}} \end{aligned}$$

Since for $|\phi_L\rangle = P_L|\phi\rangle/\|P_L|\phi\rangle\|_2$,

$$\langle\phi|M_1|\phi\rangle = g(p,L)\langle\phi_L|M_1|\phi_L\rangle = g(p,L)\,\mathrm{Tr}\left(|0\rangle\langle 0|_1 V|\phi_{\mathrm{in}}\rangle\langle\phi_{\mathrm{in}}|V^\dagger\right) \tag{66}$$

(by Equation (44), Equation (43), and Equation (42)), we thus have that if G^* is a YES instance, $\langle\phi|M_1|\phi\rangle \leq g(p,L)p_{\mathrm{bad}}$, and if G^* is a NO instance, $\langle\phi|M_1|\phi\rangle \geq g(p,L)(1-p_{\mathrm{bad}})$. Combining this with Equation (65) and Equation (64) via Hölder's inequality yields the claim. □

5.4 Applying the Lifting Lemma

We now give two examples of how to use Lemma 5.3 to obtain hardness results for APX-SIM, for the cases of $C = \mathrm{QMA}$ and $C = \mathrm{StoqMA}$.

Example 1: $C = \mathrm{QMA}$. 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 1.1) or bounded depth framework (Theorem 1.5), respectively, in conjunction with Lemma 5.3.

Theorem 5.7 (Hardness of APX-SIM for $C = \mathrm{QMA}$ via Lemma 5.3)**.** *Fix $C = \mathrm{QMA}$, 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-DAG$_s$ and C-DAG$_d$, respectively. Set $N := \min(2^{O(s(n)\log n)}, 2^{O(d(n)\log n)})$. Then, there exists a $\mathrm{poly}(N)$-time many-one reduction from G to an instance (H,a,b,δ) of APX-SIM, which satisfies: (1) H has size $\mathrm{poly}(N)$ (i.e. acts on $\mathrm{poly}(N)$ qubits/qudits, and has $\mathrm{poly}(N)$ 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) $b - a \geq 1/\mathrm{poly}(N)$ and $\delta \geq 1/\mathrm{poly}(N)$.*

Proof. If $N = 2^{O(d(n)\log n)}$, set $(G^*, f^*) = (G, \rho)$ for ρ from Lemma 4.6 and proceed to the next paragraph. Otherewise, as in Theorem 1.1, apply Lemma 2.12 to G to compute a separator tree of depth $D = O(\log(n))$ with separators of size $s = \mathrm{s}(G)$ in time $n^{O(s)}$. This is then fed into Lemma 4.7 to obtain an equivalent C-DAG G^* with $N = 2^{O(s(n)\log n)}$ nodes, each of which corresponds to a QMA verifier of size $\mathrm{poly}(N)$

(i.e. with constant promise gap, taking in proof of size $\mathrm{poly}(N)$, and running a verification circuit of size $\mathrm{poly}(N)$), along with weighting function f^*.

Next, invoke Lemma 5.3 on (G^*, f^*). Depending on whether we desire H to be 5-local on qubits or a 1D chain on qudits, set H_{w} to be Kitaev's 5-local construction [KSV02] or Hallgren, Nagaj, and Narayanaswami's 1D construction [HNN13], respectively (except in both cases, we omit the H_{out} term which penalizes rejected proofs). We then plug $H_{\mathrm{w}}(V)$ for V from Figure 4 into Lemma 5.3 with parameters as follows. (Note that by Lemma 4.7 and Lemma 4.15, V has size $\mathrm{poly}(N)$, and thus $H_{\mathrm{w}}(V)$ has size $\mathrm{poly}(N)$, by the constructions of [KSV02; HNN13].)

First, M_1 and M_2 are appropriately encoded 1-local rank-1 projectors onto $|0\rangle\langle 0|$ at the last verification time step on the output and flag qubits, respectively; thus, $\|M_2\| = 1$. The spectral gap $\Delta(H_{\mathrm{w}}(V))$ scales as $1/\mathrm{poly}(N)$ [GK12; GPY20], and $g(p, L) = 1/(1 + L) = 1/\mathrm{poly}(N)$ in both cases. If $N = 2^{O(d(n)\log n)}$, then the weighting function $W = W_{f^*}$ satisfies $W_{f^*}(G^*) \leq n(cn^{d(n)}) \in \mathrm{poly}(N)$ for any $c \in \mathrm{poly}(n)$. Else, by Lemma 4.7, the weighting function $W = W_{f^*}$ satisfies $W_{f^*}(G^*) \leq (c+1)^{O(sD)}n \in \mathrm{poly}(N)$, and $\eta \in O(1)$ (defined in Lemma 4.16, and since in time $\mathrm{poly}(N)$, each QMA verifier at a node of G^* can be amplified to have constant promise gap). In both cases, we conclude that by setting α to be a large enough fixed polynomial in N, we obtain a $1/\mathrm{poly}(N)$ promise gap in lemma 5.3, thus satisfying all claims regarding a, b, δ. All functions involved (e.g. $g(m, T)$, Δ), including the reduction itself, run in time $\mathrm{poly}(N)$. □

As noted in Section 1.1, combining Theorem 1.1 with Theorem 5.7, we have that C-DAG$_1$ can directly be embedded into an instance of APX-SIM.

Example 2: $C = \mathrm{StoqMA}$. In Lemma 5.3, when $N = 2^{O(s(n)\log n)}$ (i.e. bounded separator number framework) the promise gap of C directly influences η, which in turn affects W, α, and $\Delta(H_{\mathrm{w}}(V))$. Thus, we can apply it to obtain hardness for APX-SIM on stoquastic Hamiltonians. The tradeoff is that due to the extra log factor in Theorem 1.4 (versus Theorem 1.2), the size of the stoquastic APX-SIM instance obtained still unfortunately grows quasi-polynomially, even for $s \in O(1)$. (Recall this extra log factor is itself due to the lack of error reduction!) However, when $N = 2^{O(d(n)\log n)}$ (i.e. bounded depth framework), no such hit is incurred. As in Lemma 5.3, both frameworks are considered below.

Theorem 5.8 (Hardness of APX-SIM for $C = \mathrm{StoqMA}$ via Lemma 5.3)**.** *Fix* $C = \mathrm{StoqMA}$ *and any efficiently computable function* $s : \mathbb{N} \to \mathbb{N}$*, and define* $N := \min(2^{O(s(n)\log^2 n)}, 2^{O(d(n)\log n)})$*. Then, there exists a* $\mathrm{poly}(N)$*-time many-one reduction from any instance of* C-DAG *to an instance* (H, a, b, δ) *of APX-SIM for stoquastic* H*, which satisfies: (1)* H *has size* $\mathrm{poly}(N)$ *(i.e. acts on* $\mathrm{poly}(N)$ *qubits, and has* $\mathrm{poly}(N)$ *local terms), (2)* H *is* 2*-local, (3)* $b - a \geq 1/\mathrm{poly}(N)$ *and* $\delta \geq 1/\mathrm{poly}(N)$*.*

Proof. The proof is almost identical to that of Theorem 5.7, except for two differences: (1) Set H_{w} as the stoquastic circuit-to-Hamiltonian construction of Bravyi, Bessen, and Terhal [BBT06b], so that the output Hamiltonian H is indeed stoquastic. (Recall by that V in Figure 4 is indeed stoquastic by Remark 4.11.) (2) When $C = \mathrm{StoqMA}$ and $N = 2^{O(s(n)\log^2 n)}$, $\eta = 1/\mathrm{poly}(n)$ (versus $\eta = O(1)$), and so $c \in \mathrm{poly}(n)$. This means that although the size of G^* produced by Lemma 4.7 remains unchanged, the weighting function f^* now satisfies $W_f^* \leq 2^{O(s\log^2 n)}$ (versus $W \leq 2^{O(s\log n)}$). This, in turn, means that V (Lemma 4.15 and Figure 4) grows polynomially in size as $2^{O(s\log^2 n)}$, implying $\Delta(H_{\mathrm{w}}(V))$ (see Lemma 5 of [BBT06b]), and thus α, also scale with $2^{O(s\log^2 n)}$. (The analysis for the $N = 2^{O(d(n)\log n)}$ case remains unchanged.) □

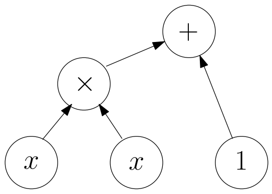


Figure 5: An example of an arithmetic circuit which computes polynomial $p(x) = x^2 + 1$.

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 $\mathrm{P}^{\mathrm{StoqMA}[\log]}$-hard [GPY20]. For clarity, this follows because $\mathrm{P}^{\mathrm{StoqMA}[\log]} = \mathrm{P}^{\|\mathrm{StoqMA}}$ [GPY20], and $\mathrm{P}^{\|\mathrm{StoqMA}}$ corresponds to a depth-1 StoqMA-DAG.

6 No-go statement for "weak compression" of polynomials

We now make a simple observation that the weighting function approach applied to NP queries (introduced in [Got95] 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"?* Throughout this section, we consider weighting functions applied to NP-DAGs.

Definitions. To define "weak compression", recall first the definition of an *arithmetic circuit*, which is a standard succinct encoding for polynomials.

Definition 6.1 (Arithmetic circuit)**.** An *arithmetic circuit* C over field F is given via a DAG as follows. Each vertex of in-degree 0 is labelled by either a variable x_i or a constant from F. Each vertex of in-degree at least 2 is labelled by either the "+" or "×" operation. Vertices of in-degree 1 are not allowed. There is a single node of out-degree 0, the *output node.* The polynomial p_C computed by C is obtained by evaluating the circuit with order of operations dictated by any topological order on C, where the output node is fixed as the last node in the order.

We now define our notion of weak compression; intuition given subsequently.

Definition 6.2 (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 exactly[28] 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 poly(m) time.
2. (Optimality preserved) For any optimal y^* maximizing $g(y^*)$ over $[0,1]^{m'}$, there exists a 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.

Very roughly, Definition 6.2 says we may efficiently reduce the number of bits required to represent the optimal value $f(x^*)$. More formally, we can efficiently map polynomial f to a new function g such that: (1) g may deviate from f arbitrarily, except on at least one optimal point x^* for f, which g must "preserve" via some g-optimal point y^*. (2) $f(y^*)$ must require fewer (i.e. B') bits than $f(x^*)$ (i.e. B) to represent.

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

Note that g is not required to be a polynomial, nor do we require that $g(y^*) = f(x^*)$.

Sanity checks regarding Definition 6.2. When $B' \geq B$, f is trivially weakly compressible to B' bits (simply set g to f). More interestly, one might ask: Given f, why can one not simply divide f by $f(x^*)$, i.e. set $f'(x) = f(x)/f(x^*)$? This would allow $B' = 1$. The problem here is that x^* is *not known a priori*, and crucially, f is specified via an arithmetic circuit. Thus, it is not at all clear how one might efficiently compute $f(x^*)$, given just this circuit description.

We now observe a no-go statement regarding weak compressibility of polynomials (expressed as arithmetic circuits).

Lemma 1.8. *Fix any function $h : R^+ \to R^+$. 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 6.2), p is weakly compressible to $h(B)$ bits. Then $\mathrm{P}^{\mathrm{NP}} \subseteq \mathrm{P}^{\mathrm{NP}[h(B)]}$.*

The proof, while simple, requires a few ingredients, and is thus given in Section 6.1. It leads to the following concrete no-go statements.

Corollary 1.9. *If any multi-linear polynomial p (represented as an arithmetic circuit) can be weakly compressed with $h(B) = O(\log B)$, then $\mathrm{P}^{\mathrm{NP}} \subseteq \mathrm{P}^{\mathrm{NP}[\log]}$.*

Proof. Immediate from Lemma 1.8 and the fact that in its proof, the admissible weighting function ω can have at most exponential total weight on an arbitrary NP-DAG, which requires B to scale as a polynomial in the worst case. □

Corollary 1.10. *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 $\mathrm{P}^{\Sigma_2^p}$).*

Proof. The proof is similar to Corollary 1.9, except when we start with a $\mathrm{P}^{\mathrm{NP}[2]}$ computation (i.e. making 2 NP queries). The weighting function ω now has at most $O(1)$ total weight, justifying the choice $B \in O(1)$ in the claim. The claim now follows since if $\mathrm{P}^{\mathrm{NP}[2]} = \mathrm{P}^{\mathrm{NP}[1]}$, then $\mathrm{PH} = \mathrm{P}^{\Sigma_2^p}$ [Har93]. □

6.1 Proof of Lemma 1.8

We first require the following lemma for encoding correct NP query strings into polynomial optimization. For concreteness, consider the admissible weighting setup of Section 4.3, specialized to the case of NP queries. Recall from Section 4.3.2 that the admissible weighting function framework allows us to reduce the task of identifying a correct NP query string $x^* \in \{0,1\}^m$ to optimizing a real-valued function $t : \{0,1\}^{\mathrm{poly}(m)} \to \mathbb{R}$ of form (c.f. Equation (25), which also allowed QMA queries)

$$t(x, y_1, \ldots, y_m) = \sum_{i=1}^{m} w_i \left(x_i V_i(x, y_i) + \frac{(1 - x_i)}{2} \right), \tag{67}$$

where w_i are the admissible weights (assumed to be rational), the bit x_i encodes the claimed answer to NP verifier V_i in the NP-DAG, and y_i is the NP proof to verifier V_i. (Remark: In Equation (25), V_i takes in $z_i(x)$ rather than all of x, where recall $z_i(x)$ selects the substring of x corresponding to the input wires of node V_i. For simplicity, here we assume without loss of generality that the function z_i is embedded into the definition of V_i itself, so that we can omit writing z_i.) Then, by Lemma 4.16, x^* is simply read off the optimal $(x^*, y_1^*, \ldots, y_m^*)$ which maximizes t.

We now use standard tricks to encode this setup into optimization of a multi-linear polynomial.

Lemma 6.3. *Let t be as in Equation (67), specified using n bits of precision (used to describe weights w_i and verifiers V_i). There exists a polynomial-time Turing machine which, given t, produces an arithmetic circuit encoding multi-linear polynomial $p_{\text{out}} : [0,1]^{\text{poly}(n)} \to \mathbb{R}^+$ with rational coefficients such that*

$$\max_{x,y_1,\ldots,y_m \in \{0,1\}^{\text{poly}(m)}} t(x, y_1, \ldots, y_m) = \max_{s \in [0,1]^{\text{poly}(m)}} p_{\text{out}}(s). \tag{68}$$

(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.

Proof. The construction applies standard tricks (used, e.g., in the proof of IP = PSPACE [Sha92]). Fix a topological ordering $R := (V_1, \ldots, V_m)$ on the vertices of the NP-DAG, and let L denote the maximum level (Definition 4.5) of any node in R. Throughout, we abuse notation and interchangably refer to V_i as both nodes in the DAG and NP verifiers V_i. The construction of p_{out} is accomplished by the following iterative algorithm:

1. Set $i = 0$.
2. While $i \le L$ do:
   (a) Let S_i denote the set of nodes at level i (with respect to R).
   (b) For all $V \in S_i$:
      i. (Map circuits to 3-SAT formulae) Map V to a 3-SAT formula ϕ_V via the Cook-Levin theorem [Coo72; Lev73a] with a minor modification: Since the input to V is *a priori* unknown (it depends on the outputs of the predecessors of V), omit the constraints in the Cook-Levin construction which force the input to a fixed string. Note:
         - $\phi_V(x, y_V, z_V)$ takes in three strings: x (query answers to predecessor queries), y_V (verifier V's proof), z_V (auxilliary variables introduced by Cook-Levin construction).
         - Without loss of generality, all ϕ_V throughout this construction are assumed to have the same number N of variables and M of clauses (via trivial padding arguments).

      ii. (Arithmetize each clause of ϕ_V) Let $c_{V,j}$ denote the jth clause of ϕ_V, where $j \in [M]$. Arithmetize each c_{V_j} via rules $\overline{x} \mapsto 1 - x$ and $x \vee y \vee z \mapsto 1 - xyz$ (with this order of precedence). For example,

$$(z_1 \vee \overline{z_2} \vee z_3) \mapsto 1 - (1 - z_1)(z_2)(1 - z_3). \tag{69}$$

      View the right hand side as a multi-linear polynomial $r_{V_j} : [0,1]^3 \to [0,1]$.

      iii. (Combine clauses of ϕ_V) For each ϕ_V, define polynomial $q_V : [0,1]^N \to [0,1]$ as $q_V := \Pi_{j=1}^M r_{V_j}$. Note q_V has range $[0,1]$, but is no longer multi-linear. Also, $\phi_V(x, y_V, z_V)$ and $q_V(x, y_V, z_V)$ take in the same arguments (although for q_V, each coordinate of x, y_V, z_V lies in $[0,1]$).

   (c) Set $i = i + 1$.
3. (Combine polynomials to simulate weighting function) Substituting into Equation (67), define polynomial

$$p(x, y_{V_1}, \ldots, y_{V_m}, z_{V_1}, \ldots, z_{V_m}) := \sum_{i=1}^m w_i \left(x_i q_{V_i}(x, y_{V_i}, z_{V_i}) + \frac{(1 - x_i)}{2} \right). \tag{70}$$

Note we define the domain of p as $[0,1]^{m(2N+1)}$; for brevity, let s_i for $i \in [m(2N+1)]$ now denote the ith real parameter of p's input. Observe p has range $[0, \sum_i |w_i|]$, since each q_{V_i} has range $[0,1]$.

4. (Linearize the polynomial) Round p to a multi-linear polynomial p_{out} via the following iterative process, for which we define $p^{(k)}$ as the polynomial p after round $k \in \{0, \ldots, m(2N+1)\}$:
   (a) Set $k = 1$.
   (b) While $k \le m(2N+1)$, set
   $$p^{(k)}(s_1, \ldots, s_{m(2N+1)}) = (1 - s_k) \cdot \text{fix}(p^{(k-1)}, k, 0) + s_k \cdot \text{fix}(p^{(k-1)}, k, 1), \tag{71}$$
   where $\text{fix}(p^{k-1}, k, b)$ is obtained by fixing s_k of p^{k-1} to $b \in \{0,1\}$.
   (c) Set $k = k + 1$.

   Observe that for all $k \in \{1, \ldots, m(2N+1)\}$,
   $$p^{(k-1)}(s_1, \ldots, s_{k-1}, b, s_{k+1}, \ldots s_{m(2N+1)}) = p^{(k)}(s_1, \ldots, s_{k-1}, b, s_{k+1}, \ldots s_{m(2N+1)}) \tag{72}$$
   for any $b \in \{0,1\}$. Thus, $p_{\text{out}} := p^{(m(2N+1))}$ satisfies $p_{\text{out}}(s) = p(s)$ for all $s \in \{0,1\}^{m(2N+1)}$. Moreover, by construction p_{out} is multi-linear and has range $[0, \sum_i |w_i|]$ (since each iteration of line 4b introduces a convex combination over local assignments).

Finally, we assume all arithmetic operations above are represented implicitly via gates of an arithmetic circuit (required due to Step 2biii, as expanding q_V explicitly in a monomial basis can result in exponentially many terms). The resulting arithmetic circuit clearly has size $\text{poly}(n)$.

Correctness. Since p_{out} is multi-linear, it obtains[29] its maximum on an extreme point of the compact set $[0,1]^{m(2N+1)}$, i.e.
$$\max_{s \in [0,1]^{m(2N+1)}} p_{\text{out}}(s) = \max_{s \in \{0,1\}^{m(2N+1)}} p_{\text{out}}(s). \tag{73}$$

Thus, we may restrict attention[30] to $s \in \{0,1\}^{m(2N+1)}$. But on this set, ϕ_V (Cook-Levin output) and q_V (arithmetization of ϕ_V) coincide. We conclude
$$\max_{s \in [0,1]^{m(2N+1)}} p_{\text{out}}(s) = \max_{x, y_1, \ldots, y_m \in \{0,1\}^{\text{poly}(m)}} t(x, y_1, \ldots, y_m) \tag{74}$$

for t from Equation (67). Moreover, recalling that $s = x y_{V_1} \cdots y_{V_m} z_{V_1} \cdots z_{V_m}$ (viewed as a concatenation of strings), it follows that given the optimal s^*, we may recover the correct NP query string simply by reading off x. □

With Lemma 6.3 in hand, the proof of Lemma 1.8 now follows straightforwardly.

Proof of Lemma 1.8. The proof is similar to that of Theorem 1.1 (and thus Theorem 4.1), except we need not apply the Compression Lemma (Lemma 4.7) in that the content of Section 4.3 suffices, and now

[29] Here is a simple proof via exchange argument, for completeness: Let $x = (x_1, ..., x_n)$ be a point maximizing multi-linear $f : [a,b]^n \to \mathbb{R}$ for arbitrary $a, b \in \mathbb{R}$. Assume without loss of generality $x_1 \notin \{a, b\}$. Then, fixing $x_2, \ldots, x_n$, the resulting function $f(x_1)$ is linear in x_1 by definition, and so $\max(f(a), f(b)) \ge f(x_1)$. Exchanging $\arg\max(f(a), f(b))$ for x_1 completes the claim.

[30] Note the linearization of Step 4 is *necessary* to obtain this statement. For example, consider an unsatisfiable 2-SAT formula $\phi(x_1, x_2) = (x_1 \vee x_2) \wedge (\overline{x_1} \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_2})$. Let q be the multi-variate polynomial produced by arithmetizing ϕ as in steps 2(*ii*) and 2(*iii*). Then, the maximum value of q over strings is 0, but setting each variable to 1/2 yields value $(3/4)^4 > 0$. With this said, note that for 3-SAT, since a 7/8-approximation ratio is optimal via the PCP theorem [Hås97], one can show via AM-GM inequality that for any unsatisfiable ϕ, optimizing all variables of q over $[0,1]$ yields value at most $(7/8)^m$ for m clauses. Thus, up to inverse exponential corrections, one *could* avoid the linearization step, but the tradeoff is added clutter and the need to assume P $\neq$ NP.

we use Lemma 6.3 to make the connection to polynomials. Specifically, let Π be any instance of a P^{NP} problem, and M a P^{NP} machine deciding Π. Map the NP-DAG representing M's action directly (i.e. without utilizing Lemma 4.7) to the function t in Equation (67). For this, the weights w_i can be any c-admissible weighting function which satisfies the preconditions of Lemma 4.16; for concreteness, choose the 2-admissible function $\omega(v) = 3^{|\mathrm{Desc}(v)|}$ from Lemma 4.6. Apply Lemma 6.3 to map t to polynomial $p_{\mathrm{out}}(x, y_1, \ldots, y_m, z_1, \ldots, z_m)$. Since p_{out} is efficiently evaluated on any given input, and has an optimal value $p_{\mathrm{out}}(x^*, y_1^*, \ldots, y_m^*, z_1^*, \ldots, z_m^*)$ expressible using B bits of precision[31], a binary search using B queries to an NP-oracle suffices to identify an optimal input $(x^*, y_1^*, \ldots, y_m^*, z_1^*, \ldots, z_m^*)$. By Lemma 6.3, one can now efficiently extract the answers to all NP queries made by M (specifically, this is the string x^*), and thus efficiently simulate M itself to decide Π. □

Acknowledgments

We thank Eric Allender, Johannes Bausch, Stephen Piddock, James Watson, and Justin Yirka for helpful discussions. SG acknowledges funding from DFG grants 432788384 and 450041824.

References

[AGL20] D. Aharonov, A. B. Grilo, and Y. Liu. "StoqMA vs. MA: the power of error reduction." In: *arXiv:2010.02835 [quant-ph]* (Oct. 2020). arXiv: 2010.02835.

[AI] D. Aharonov and S. Irani. "Hamiltonian Complexity in the Thermodynamic Limit." Available at arXiv.org quant-ph/2107.06201.

[Amb14] A. Ambainis. "On Physical Problems that are Slightly More Difficult than QMA." In: *2014 IEEE 29th Conference on Computational Complexity (CCC)*. 2014, pp. 32–43.

[Ami10] E. Amir. "Approximation Algorithms for Treewidth." In: *Algorithmica* 56.4 (Apr. 2010), pp. 448–479. ISSN: 1432-0541. DOI: 10.1007/s00453-008-9180-4.

[BBT06a] S. Bravyi, A. J. Bessen, and B. M. Terhal. "Merlin-Arthur Games and Stoquastic Complexity." In: (Dec. 2006). arXiv: quant-ph/0611021 [quant-ph].

[BBT06b] S. Bravyi, A. J. Bessen, and B. M. Terhal. "Merlin-Arthur Games and Stoquastic Complexity." In: *arXiv e-prints*, quant-ph/0611021 (Nov. 2006), quant–ph/0611021. arXiv: quant-ph/0611021 [quant-ph].

[Bei91] R. Beigel. "Bounded queries to SAT and the Boolean hierarchy." In: *Theoretical Computer Science* 84.2 (1991), pp. 199–223. ISSN: 0304-3975. DOI: http://dx.doi.org/10.1016/0304-3975(91)90160-4.

[BH91] S. R. Buss and L. Hay. "On truth-table reducibility to SAT." In: *Information and Computation* 91.1 (1991), pp. 86–102. ISSN: 0890-5401. DOI: 10.1016/0890-5401(91)90075-D.

[BHS80] J. L. Bentley, D. Haken, and J. B. Saxe. "A General Method for Solving Divide-and-Conquer Recurrences." In: *SIGACT News* 12.3 (Sept. 1980), pp. 36–44. ISSN: 0163-5700. DOI: 10.1145/1008861.1008865.

[BHW89] R. Beigel, L. Hemachandra, and G. Wechsung. "On the power of probabilistic polynomial time: $\mathrm{P}^{\mathrm{NP}[\log]} \subseteq \mathrm{PP}$." In: *[1989] Proceedings. Structure in Complexity Theory Fourth Annual Conference*. June 1989, pp. 225–227. DOI: 10.1109/SCT.1989.41828.

[31] For an arbitrary NP-DAG, B can be polynomial in the NP-DAG's size.

[Bod+13] H. L. Bodlaender, P. G. Drange, M. S. Dregi, F. V. Fomin, D. Lokshtanov, and M. Pilipczuk. "An $O(c^k n)$ 5-Approximation Algorithm for Treewidth." In: *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. 2013, pp. 499–508. DOI: 10.1109/FOCS.2013.60.

[Bod+95] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks. "Approximating Treewidth, Pathwidth, Frontsize, and Shortest Elimination Tree." In: *J. Algorithms* 18.2 (Mar. 1995), pp. 238–255. ISSN: 0196-6774. DOI: 10.1006/jagm.1995.1009.

[Bod93] H. L. Bodlaender. "A linear time algorithm for finding tree-decompositions of small treewidth." In: *Proceedings of the twenty-fifth annual ACM symposium on Theory of Computing*. STOC '93. San Diego, California, USA: Association for Computing Machinery, June 1993, pp. 226–234. ISBN: 9780897915915. DOI: 10.1145/167088.167161.

[Boo14] A. D. Bookatz. "QMA-complete problems." In: *Quantum Information & Computation* 14.5&6 (Apr. 2014), pp. 361–383. ISSN: 1533-7146.

[CM16] T. Cubitt and A. Montanaro. "Complexity classification of local Hamiltonian problems." In: 45.2 (2016), pp. 268–316.

[Coo71] S. A. Cook. "The complexity of theorem-proving procedures." In: *Proceedings of the third annual ACM symposium on Theory of computing*. STOC '71. Shaker Heights, Ohio, USA: Association for Computing Machinery, May 1971, pp. 151–158. ISBN: 9781450374644. DOI: 10.1145/800157.805047.

[Coo72] S. Cook. "The complexity of theorem proving procedures." In: *3rd ACM Symposium on Theory of Computing (STOC 1972)*. 1972, pp. 151–158.

[CS12] A. Chailloux and O. Sattath. "The Complexity of the Separable Hamiltonian Problem." In: *2012 IEEE 27th Conference on Computational Complexity*. ISSN: 1093-0159. June 2012, pp. 32–41. DOI: 10.1109/CCC.2012.42.

[CS92] J. Castro and C. Seara. "Characterizations of some complexity classes between Ω_2^p and Δ_2^p." In: *STACS 92*. Ed. by A. Finkel and M. Jantzen. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1992, pp. 303–317. ISBN: 9783540467755. DOI: 10.1007/3-540-55210-3_192.

[D A+09] D. Aharonov, D. Gottesman, S. Irani, and J. Kempe. "The Power of Quantum Systems on a Line." In: *Communications in Mathematical Physics* 287 (2009), pp. 41–65.

[DS09] D. Gottesman and S. Irani. "The Quantum and classical complexity of translationally invariant tiling and Hamiltonian problems." In: *50th IEEE Symposium on Foundations of Computer Science (FOCS 2009)*. 2009, pp. 95–104.

[FL16] B. Fefferman and C. Lin. "Quantum Merlin Arthur with Exponentially Small Gap." In: (Jan. 2016). arXiv: 1601.01975 [quant-ph].

[Gha+15] S. Gharibian, Y. Huang, Z. Landau, and S. W. Shin. "Quantum Hamiltonian Complexity." In: *Foundations and Trends® in Theoretical Computer Science* 10.3 (Oct. 2015), pp. 159–282. ISSN: 1551-305X. DOI: 10.1561/0400000066.

[GJ79] M. R. Garey and D. S. Johnson. *COMPUTERS and INTRACTABILITY: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[GK12] S. Gharibian and J. Kempe. "Hardness of approximation for quantum problems." In: *39th International Colloquium on Automata, Languages and Programming (ICALP)*. 2012, pp. 387–398.

[Got95] G. Gottlob. "NP trees and Carnap's modal logic." In: *Journal of the ACM* 42.2 (Mar. 1995), pp. 421–457. ISSN: 0004-5411. DOI: 10.1145/201019.201031.

[GPY20] S. Gharibian, S. Piddock, and J. Yirka. "Oracle Complexity Classes and Local Measurements on Physical Hamiltonians." In: *37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*. Ed. by C. Paul and M. Bläser. Vol. 154. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 20:1–20:37. ISBN: 978-3-95977-140-5. DOI: 10.4230/LIPIcs.STACS.2020.20. arXiv: 1909.05981 [quant-ph].

[Gru12] H. Gruber. "On balanced separators, treewidth, and cycle rank." In: *Journal of Combinatorics* 3.4 (2012), pp. 669–681. ISSN: 2150-959X. DOI: 10.4310/JOC.2012.v3.n4.a5.

[GY19] S. Gharibian and J. Yirka. "The complexity of simulating local measurements on quantum systems." In: *Quantum* 3 (Sept. 2019), p. 189. DOI: 10.22331/q-2019-09-30-189.

[Har93] J. Hartmanis. "Sparse Complete Sets for NP and the Optimal Collapse of the Polynomial Hierarchy." In: *Current Trends in Theoretical Computer Science*. WORLD SCIENTIFIC, June 1993, pp. 403–411. DOI: 10.1142/9789812794499_0029.

[Hås97] J. Håstad. "Some optimal inapproximability results." In: *29th Symposium on Theory of Computing (STOC 1997)*. 1997, pp. 1–10.

[Hem89] L. A. Hemachandra. "The strong exponential hierarchy collapses." In: *Journal of Computer and System Sciences* 39.3 (Dec. 1989), pp. 299–322. ISSN: 0022-0000. DOI: 10.1016/0022-0000(89)90025-1.

[HHR97] E. Hemaspaandra, L. A. Hemaspaandra, and J. Rothe. "Exact analysis of Dodgson elections: Lewis Carroll's 1876 voting system is complete for parallel access to NP." In: *Automata, Languages and Programming*. Ed. by P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1997, pp. 214–224. ISBN: 9783540691945. DOI: 10.1007/3-540-63165-8_179.

[HM13] A. W. Harrow and A. Montanaro. "Testing product states, quantum Merlin-Arthur games and tensor optimisation." In: *Journal of the ACM* 60.1 (Feb. 2013), pp. 1–43. ISSN: 0004-5411, 1557-735X. DOI: 10.1145/2432622.2432625. arXiv: 1001.0017.

[HNN13] S. Hallgren, D. Nagaj, and S. Narayanaswami. "The Local Hamiltonian problem on a line with eight states is QMA-complete." In: *Quantum Information & Computation* 13.9&10 (2013), pp. 0721–0750.

[Kar72] R. M. Karp. "Reducibility among Combinatorial Problems." In: ed. by R. E. Miller, J. W. Thatcher, and J. D. Bohlinger. The IBM Research Symposia Series. Boston, MA: Springer US, 1972, pp. 85–103. ISBN: 9781468420012. DOI: 10.1007/978-1-4684-2001-2_9.

[KKR06] J. Kempe, A. Kitaev, and O. Regev. "The Complexity of the Local Hamiltonian Problem." In: *SIAM Journal on Computing* 35.5 (Jan. 2006), pp. 1070–1097. DOI: 10.1137/s0097539704445226.

[KR03] J. Kempe and O. Regev. "3-local Hamitonian is QMA-complete." In: *Quantum Information & Computation* 3.3 (May 2003), pp. 258–264. ISSN: 1533-7146.

[Kre88a] M. W. Krentel. "The complexity of optimization problems." In: *Journal of Computer and System Sciences* 36.3 (1988), pp. 490–509. ISSN: 0022-0000. DOI: http://dx.doi.org/10.1016/0022-0000(88)90039-6.

[Kre88b] M. W. Krentel. “The complexity of optimization problems.” In: *Journal of Computer and System Sciences* 36.3 (1988), pp. 490–509. ISSN: 0022-0000. DOI: 10.1016/0022-0000(88)90039-6.

[Kre92] M. W. Krentel. “Generalizations of Opt P to the polynomial hierarchy.” In: *Theoretical Computer Science* 97.2 (Apr. 1992), pp. 183–198. ISSN: 0304-3975. DOI: 10.1016/0304-3975(92)90073-O.

[KSV02] A. Y. Kitaev, A. H. Shen, and M. N. Vyalyi. *Classical and Quantum Computation.* USA: American Mathematical Society, 2002. ISBN: 0821832298.

[Lev73a] L. Levin. “Universal search problems.” In: *Problems of Information Transmission* 9.3 (1973), pp. 265–266.

[Lev73b] L. A. Levin. “Universal sequential search problems.” In: *Problems of Information Transmission* 9.3 (1973), pp. 265–266.

[O G06] O. Goldreich. “On promise problems: A survey.” In: *Theoretical Computer Science* 3895 (2006), pp. 254–290.

[Osb12] T. J. Osborne. “Hamiltonian complexity.” In: *Reports on Progress in Physics* 75.2 (Jan. 2012), p. 022001. ISSN: 0034-4885. DOI: 10.1088/0034-4885/75/2/022001.

[Pap82] C. H. Papadimitriou. “On the complexity of unique solutions.” In: *23rd Annual Symposium on Foundations of Computer Science (SFCS 1982).* 1982, pp. 14–20. DOI: 10.1109/SFCS.1982.28.

[PZ82] C. H. Papadimitriou and S. K. Zachos. “Two remarks on the power of counting.” In: *Theoretical Computer Science.* Ed. by A. B. Cremers and H.-P. Kriegel. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1982, pp. 269–275. ISBN: 9783540394211. DOI: 10.1007/BFb0036487.

[Ree92] B. A. Reed. “Finding approximate separators and computing tree width quickly.” In: *Proceedings of the twenty-fourth annual ACM symposium on Theory of Computing.* STOC ’92. Victoria, British Columbia, Canada: Association for Computing Machinery, July 1992, pp. 221–228. ISBN: 9780897915113. DOI: 10.1145/129712.129734.

[RS86] N. Robertson and P. Seymour. “Graph minors. II. Algorithmic aspects of tree-width.” In: *Journal of Algorithms* 7.3 (1986), pp. 309–322. ISSN: 0196-6774. DOI: https://doi.org/10.1016/0196-6774(86)90023-4.

[RT19] R. Raz and A. Tal. “Oracle Separation of BQP and PH.” In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing.* STOC 2019. Phoenix, AZ, USA: Association for Computing Machinery, 2019, pp. 13–23. ISBN: 9781450367059. DOI: 10.1145/3313276.3316315.

[S A10] S. Aaronson. “BQP and the polynomial hierarchy.” In: *42nd ACM Symposium on the Theory of Computing (STOC 2010).* 2010, pp. 141–150.

[Sch03] P. Schnoebelen. “Oracle Circuits for Branching-Time Model Checking.” In: *Automata, Languages and Programming: 30th International Colloquium, ICALP 2003 Eindhoven, The Netherlands, June 30 – July 4, 2003 Proceedings.* Ed. by J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 790–801. ISBN: 978-3-540-45061-0. DOI: 10.1007/3-540-45061-0_62.

[Sha92] A. Shamir. “IP = PSPACE.” In: *J. ACM* 39.4 (Oct. 1992), pp. 869–877. ISSN: 0004-5411. DOI: 10.1145/146585.146609.

[Sto76] L. J. Stockmeyer. “The polynomial-time hierarchy.” In: *Theoretical Computer Science* 3.1 (Oct. 1976), pp. 1–22. ISSN: 0304-3975. DOI: 10.1016/0304-3975(76)90061-X.

[Wag87] K. W. Wagner. “More complicated questions about maxima and minima, and some closures of NP.” In: *Theoretical Computer Science* 51.1 (Jan. 1987), pp. 53–80. ISSN: 0304-3975. DOI: 10.1016/0304-3975(87)90049-1.

[Wag88] K. Wagner. “Bounded query computations.” In: *[1988] Proceedings. Structure in Complexity Theory Third Annual Conference* (1988). DOI: 10.1109/SCT.1988.5286.

[WB21] J. D. Watson and J. Bausch. “The Complexity of Approximating Critical Points of Quantum Phase Transitions.” Available at arXiv.org quant-ph/2105.13350. 2021.

[WBG20] J. Watson, J. Bausch, and S. Gharibian. “The Complexity of Translationally Invariant Problems beyond Ground State Energies.” Available at arXiv.org quant-ph/2012.12717. 2020.

[WC21] J. D. Watson and T. S. Cubitt. “Computational Complexity of the Ground State Energy Density Problem.” Available at arXiv.org quant-ph/2107.05060. 2021.