Electronic Colloquium on Computational Complexity, Report No. 43 (2020)

A combinatorial MA-complete problem

Dorit Aharonov[1] and Alex B. Grilo[2]

[1]Hebrew University of Jerusalem, Jerusalem, Israel
[2]CWI and QuSoft, Amsterdam, The Netherlands

Abstract

Despite the interest in the complexity class MA, the randomized analog of NP, there is just a couple of known natural (promise-)MA-complete problems, the first due to Bravyi and Terhal (SIAM Journal of Computing 2009) and the second due to Bravyi (Quantum Information and Computation 2015). Surprisingly, both problems are stated using terminology from quantum computation. This fact makes it hard for *classical* Complexity Theorists to study these problems, and prevents possible progress, e.g., on the important question of derandomizing MA.

In this note we define a natural combinatorial problem called SetCSP and prove its MA-completeness. The problem generalizes the constraint satisfaction problem (CSP) into constraints on *sets* of strings.

This note is in fact a combination of previously known works: the brilliant work of Bravyi and Terhal, together with an observation made in our previous work (Aharonov and Grilo, FOCS 2019) that a restricted case of the Bravyi and Terhal's MA complete problem (namely, the uniform case) is already complete, and moreover, that this restricted case can be stated using a classical, combinatorial description. Here we flesh out this observation.

This note, along with a translation of the main result of Aharonov and Grilo to the SetCSP language, implies that finding a gap-amplification procedure for SetCSP problems (namely a generalization to SetCSPs of the gap-amplification used in Dinur's PCP proof) would imply MA=NP. This would provide a resolution of the major problem of derandomizing MA; in fact, the problem of finding gap-amplification for SetCSP is in fact equivalent to the MA=NP problem.

1 Introduction

The complexity class MA is a natural extension of NP proof systems to the probabilistic setting [Bab85]. There is a lot of evidence towards the fact that these two complexity classes are actually equal [IKW02, GZ11, HILL99, BFNW93, NW94, IW97, STV01, KI04], but it is an open question if every problem in MA can be solved even in *non-deterministic* sub-exponential time.

Surprisingly, the very first *natural* MA-complete[1] problem, found by Bravyi and Terhal [BT09] only close to 25 years after the definition of the class (!), is defined using *quantum* terminology. Though why would randomized NP have anything to do with quantum? Bravyi and Terhal show

[1] For PromiseMA, it is folklore that one can define complete problems by extending NP-complete problems (see, e.g. [SW]): we define an exponential family of 3SAT formulas (given as input succinctly) and we have to decide if there is an assignment that satisfies all of the formulas, or for every assignment, a random formula in the family will not be satisfied with good probability.

ISSN 1433-8092

that deciding if a stoquastic k-Local Hamiltonian[2]is frustration-free or inverse polynomially frustrated is promise-MA-complete. Later, Bravyi [Bra14] proved MA-completeness of yet another quantum Hamiltonian problem.

This leaves us in a situation in which the only natural MA complete problems known are stated and proved in quantum computational complexity language. Natural complete problems for the class MA might enable access to the major open problem of derandomization of MA; and possibly to other related problems such as PCP for MA [AG19][3]. However, though the problems proposed in [BT09, Bra14] are very natural within quantum complexity theory, the fact that they are defined within the area of quantum computation seems to pose a language and conceptual barrier that might prevent progress on them, and make it hard for *classical* complexity theorists to study them.

The goal of this note is to provide a classically, combinatorially-defined complete problem for MA. The problem we suggest is based on the problem of [BT09]; in order to turn it into a classically stated problem, one needs to rely on a simple observation made in [AG19] which says that the problem of [BT09] remains promise-MA complete even when restricted to what is referred to in [AG19] as *uniform* stoquastic Hamiltonians. From that point, almost everything we prove in this note is simply translating into classical language results and observations given in [BT09]. We present the results from scratch avoiding any quantum notation or quantum jargon what-so-ever. We mention that this note can be viewed as a fleshing out of the hint given in [AG19], of how to extract from [BT09] a *combinatorial* MA-complete problem.

The new MA complete problem, called the *Set-Constraint Satisfaction Problem*, or SetCSP, is a generalization of the standard *Constraint Satisfaction Problem (*CSP*)*, except we consider constraints that act on *sets* of strings instead of on a single string - we call those *set-constraints* (see Section 3 for the exact definition of a set-constraint.). The input to the SetCSP problem is a collection of such set-constraints, and the output is whether there is a set of strings S that satisfies all set-constraints in this collection, or any S is ε-far from satisfying this set-constraint collection (see Definitions 3.2 and 3.9 for formal definitions of "satisfying a set-constraint" and "far"). We denote this problem by $\mathsf{SetCSP}_\varepsilon$. Following the ideas of [BT09, KSV02], we then show that there exists some inverse polynomial function ε, such that this problem is MA-complete. We stress that the definition of the problem and the proof of MA-completeness only use standard combinatorial concepts from set- and graph-theory.

We hope that this definition leads to further understanding of the class MA and the MA vs. NP question.

1.1 Future work and open questions

In a similar way to what is done in this note, it turns out that it is possible to translate several other results from Hamiltonian complexity to the SetCSP language; this brings us to a very interesting conclusion.

First, by following the lines of [AvDK$^+$08], it can be proved that SetCSP with set-constraints arranged on a $2D$-lattice is still MA-hard. This means that the problem remains MA-hard even when each bit participates in $O(1)$ set-constraints, and each set-constrain acts on $O(1)$ bits.

[2]Stoquastic Hamiltonians sit between classical Hamiltonians (CSPs) and general quantum Hamiltonians.

[3]We notice that Drucker [Dru11] proves a PCP theorem for AM; in the definition he uses for AM, the coins are public and the prover sees them (See Section 2.3 in [Dru11]); but his result does not hold when the coins are private, namely for MA.

In addition, we claim that the result of [AG19] can also be translated to SetCSP language, implying that the *gapped* version of the same SetCSP problem is in NP. More precisely, in the previous notation, the problem of SetCSP when ε is a constant, the locality k (number of bits in a set-constraint) and degree d (number of set-constraints each bit participates in) are bounded from above by a constant, is in NP.

We leave writing up the details for these two translations to the SetCSP language to future work, but we highlight that together they lead to a very important and surprising equivalence[4]:

Proposition: There is a gap amplification reduction [5] for SetCSP, if and only if MA=NP.

We note that it is easy to see that MA=NP implies such gap-amplification for SetCSP, by the PCP theorem[Din07]; it is the other direction of deriving MA=NP from gap-amplification for SetCSP, that is the new contribution. This implication suggests a new path to the major open problem of derandomizing MA.

Organization

We define some notation and recall basic concepts and results of complexity theory in Section 2. Then we define the Set-Constraint Satisfaction Problem in Section 3, and then we show its MA-completeness in Section 4. We remind some standard concepts of graph theory which are required for our main proof, in Appendix A.

2 Preliminaries

2.1 Notation

For $n \in \mathbb{N}^+$, we denote $[n] = \{0, ..., n-1\}$. For any n-bit string, we index its bits from 0 to $n-1$. For $x \in \{0,1\}^n$ and $J \subseteq [n]$, we denote $x|_J$ as the substring of x on the positions contained in J. For $x \in \{0,1\}^{|J|}$, $y = \{0,1\}^{n-|J|}$ and $J \subseteq [n]$, we define $x^J y^{\overline{J}}$ to be the unique n-bit string w such that $w|_J = x$ and $w|_{\overline{J}} = y$, where $\overline{J} = [n] \setminus J$. For two strings, x and y, we denote $x \,||\, y$ their concatenation. For a string of bits x, $|x|$ denotes the number of bits in x.

2.2 Complexity classes

A (promise) problem $A = (A_{yes}, A_{no})$ consists of two non-intersecting sets $A_{yes}, A_{no} \subseteq \{0,1\}^*$. We define now the main complexity classes that are considered in this work.

Definition 2.1 (NP)**.** A problem $A = (A_{\text{yes}}, A_{\text{no}})$ is in NP if and only if there exist two polynomials p, q and a deterministic algorithm D, where D takes as input a string $x \in \Sigma^*$ and a $p(|x|)$-bit witness y, runs in time $q(|x|)$ and outputs a bit 1 (accept) or 0 (reject) such that:
Completeness. If $x \in A_{\text{yes}}$, then there exists a witness y such that D outputs accept on (x, y).
Soundness. If $x \in A_{\text{no}}$, then for any witness y, D outputs reject on (x, y).

We can then generalize this notion, by giving the verification algorithm the power to flip random coins, leading to the complexity class MA.

[4]This equivalence was highlighted in [AG19] in a quantum language
[5]In the same sense as Dinur's gap amplification reduction for CSP[Din07]

Definition 2.2 (MA)**.** A (promise) problem $A = (A_{\text{yes}}, A_{\text{no}})$ is in MA if and only if there exist two polynomials p, q and a probabilistic algorithm R, where R takes as input a string $x \in \Sigma^*$ and a $p(|x|)$-bit witness y, runs in time $q(|x|)$ and decides on acceptance or rejection of x such that:
Completeness. If $x \in A_{\text{yes}}$, then there exists a witness y such that R accepts (x, y) with probability 1.
Soundness. If $x \in A_{\text{no}}$, then for any witness y, R accepts (x, y) with probability at most $\frac{1}{3}$.

The standard definition of MA [Bab85] requires yes-instances to be accepted with probability at least $\frac{2}{3}$, but it has been shown that there is no change in the computational power if we require the verification algorithm to always accept yes-instances [ZF87, GZ11].

2.3 Reversible circuits

In classical complexity theory, we usually describe the verification procedure for NP and MA as algorithms. Such polynomial time algorithms can be converted into a uniform family of polynomial-size circuits, which usually are described using AND, OR and NOT gates. As we will see later, these circuits are not suitable in our context because we need our circuits to be *reversible*, namely made of reversible gates. By this we mean that for every output of some gate, it should be possible to compute the corresponding input by applying the *inverse* of this gate. Clearly, we can also invert the entire circuit made of such gates by running all inverses of the gates in reverse order.

Circuits can be made reversible by using the *universal* gateset $\{\text{NOT}, \text{CNOT}, \text{CCNOT}\}$,[6] and additional auxiliary bits initialized to 0 are provided. It is folklore that the overhead for this conversion is linear in the number of gates. We will assume all verifiers considered are reversible.

For randomized circuits, we can also assume the circuit uses only reversible gates, in the following way: we assume that the random bits are provided as part of the initial string on which the circuit acts; we can view it as part of the input. The rest of the circuit is reversible.

For decision problems, we use the convention that the first bit is used as the output of the circuit, namely, accept or reject.

Finally, let $G \in \{\text{NOT}, \text{CNOT}, \text{CCNOT}\}$ be a gate to be applied on the set of qubits J out of some n-bit input x. We slightly abuse notation (but make it much shorter!) and denote $G(x) \overset{\text{def}}{=} x|_{\overline{J}}^{\overline{J}} G(z|_J)^J$. Namely, we understand the action of the k-bit gate G on an $n > k$ bit string x by applying G only on the relevant bits, and leaving all other bits intact.

3 Set-Constraint Satisfaction Problem

The goal of this section is to present the central problem of this note, the Set Constraint Satisfaction Problem (SetCSP).

We start by recalling the standard Constraint Satisfaction Problem (CSP). We choose to present CSP in a way which is more adapted to our generalization (but still equivalent to the standard definition). An instance of k-CSP is a sequence of constraints $C_1, ..., C_m$. In this paper, we see each constraint C_i as a tuple $(J(C_i), Y(C_i))$, where $J(C_i) \subseteq [n]$ is a subset of at most k distinct elements of $[n]$ (these are referred to as "the bits on which the constraint acts") and $Y(C_i)$ is a subset of

[6]NOT is the 1-bit gate that on input a outputs $1 \oplus a$; CNOT is the 2-bit gate that on input (a, b) outputs $(a, b \oplus a)$; finally CCNOT is the 3-bit gate that on input (a, b, c) outputs $(a, b, c \oplus ac)$, i.e., it flips the last bit iff $a = b = 1$. Notice that $\{\text{NOT}, \text{CCNOT}\}$ is already universal for classical computation and we just add CNOT for convenience.

k-bit strings, namely $Y(C_i) \subseteq \{0,1\}^{|J(C_i)|}$ (these are called the "allowed strings"). We say that a string $x \in \{0,1\}^n$ *satisfies* C_i if $x|_{J(C_i)} \in Y(C_i)$. The familiar problem of k-CSP, in this notation, is to decide whether there exists an n-bit string x which satisfies C_i, for all $1 \leq i \leq m$.

In k-SetCSP, our generalization of k-CSP, the constraints C_i are replaced by what we call "set-constraints" that are satisfied by (or alternatively, that *allow*), *sets* of strings $S \subseteq \{0,1\}^n$. The most important definitions in this note are the following two.

Definition 3.1 (Set constraints)**.** A k-local set-constraint C consists of a) a tuple of k distinct elements of $[n]$, denoted $J(C)$ (we have $|J(C)| = k$, and we refer to $J(C)$ as the bits which the set-constraint C acts on), and b) a collection $Y(C) = \{Y_1, ..., Y_\ell\}$ where $Y_i \subseteq \{0,1\}^k$ and $Y_i \cap Y_j = \emptyset$ for all distinct $1 \leq i, j \leq \ell$.

Definition 3.2 (A set of strings satisfying a constraint)**.** We say that a set of strings $S \subseteq \{0,1\}^n$ *satisfies* the k-local set-constraint C if first, the k-bit restriction of any string in S to $J(C)$ is contained in one of the sets, i.e., for all $x \in S$, $x|_{J(C)} \in \bigcup_j Y_j$. Secondly we require that if $x \in S$ and $x|_{J(C)} \in Y_j$, then for every y such that $y|_{J(C)} \in Y_j$ and $y|_{\overline{J(C)}} = x|_{\overline{J(C)}}$, then $y \in S$. In other words, all elements of the subset Y_j appear together in S (with the same extension to the remaining bits) or none of them appears.

An instance of k-SetCSP consists of m such k-local set-constraints, and we ask if there is some non-empty $S \subseteq \{0,1\}^n$ that satisfies each of the set constraints, or if all sets of n-bit strings are *far* from satisfying them. How to define *far*? We quantify the distance from satisfaction using a generalization of the familiar notation of unsat from PCP theory [Din07]; we denote the generalized notion by set-unsat$(\mathcal{C}, S)$. Intuitively, this quantity captures how much we need to modify S in order to satisfy the set-constraints. Making this definition more precise will require some work; However we believe it already makes some sense intuitively, so we present the definition of the SetCSP problem now, and then state our main result. The exact definition of set-unsat$(\mathcal{C}, S)$ is given in Section 3.1.

Definition 3.3 (k-local Set Constraint Satisfaction problem (k-SetCSP$_\varepsilon$))**.** Fix the two constants $d, k \in \mathbb{N}^+$, as well as a monotone function $\varepsilon : \mathbb{N}^+ \to (0,1)$ to be some parameters of the problem. An instance to the *k-local Set Constraint Satisfaction problem* is a sequence of $m(n)$ k-local set-constraints $\mathcal{C} = (C_1, ..., C_m)$ on $\{0,1\}^n$, where m is some polynomial in n. Under the promise that one of the following two holds, decide whether:
Yes. There exists some $S \subseteq \{0,1\}^n$ such that set-unsat$(\mathcal{C}, S) = 0$
No. For all $S \subseteq \{0,1\}^n$, set-unsat$(\mathcal{C}, S) \geq \varepsilon(n)$.

We can now state the main theorem in this note:

Theorem 3.4. *There exists some $\epsilon = 1/poly(n)$ function, such that k-SetCSP$_\varepsilon$ is promise-*MA *complete, for $k \geq 6$.*

3.1 Satisfiability, frustration and the definition of set-unsat$(\mathcal{C}, S)$.

We present some concepts required for the formal definition.

Definition 3.5 (Neighboring strings)**.** Let C be some set-constraint. Two distinct strings x and y are said to be C-neighbors if $x|_{\overline{J(C)}} = y|_{\overline{J(C)}}$ and $x|_{J(C)}, y|_{J(C)} \in Y_i$, for some $Y_i \in Y(C)$. We call a string x a C-neighbor of S if there exists a string $y \in S$ such that x is a $C-$neighbor of y.

We also define the C-longing strings in a set of strings S: these are the strings that are in S but are C-neighbors of some string that is not in S.

Definition 3.6 (Longing string)**.** Given some set $S \subseteq \{0,1\}^n$ and a set-constraint C, $x \in S$ is a C-longing string if x is a C-neighbor of some $y \notin S$.

A useful definition is that of bad strings for some set-constraint C, which in short are the strings that do not appear in any subset of $Y(C)$.

Definition 3.7 (C-Bad string)**.** Given a set-constraint C, with $Y(C) = \{Y_1, ...Y_\ell\}$, a string $x \in \{0,1\}^n$ is C-bad if $x|_{J(C)} \notin \cup_i Y_i$. We abuse the notation and whenever x is C-bad, we say $x \notin Y(C)$.

The following complementary definition will be useful:

Definition 3.8 (Good string)**.** We say that a string is C-good if it is not C-bad. We say that it is a "good string for the set-constraint collection $\mathcal{C}$" if it is C-good for all set-constraints $C \in \mathcal{C}$. When the collection $\mathcal{C}$ is clear from context (as it is throughout this note), we omit mentioning of the set-constraints collection and just say that the string is "good".

Given Definitions 3.6 and 3.7 above, it is easy to see that a set of strings $S \subseteq \{0,1\}^n$ satisfies a set-constraint C (by Definition 3.2) iff S contains no C-bad strings and no C-longing strings.

We now provide a way to quantify how far S is from satisfying C.

Definition 3.9 (Satisfiability of set-constraints)**.** Let $S \subseteq \{0,1\}^n$ and C be a k-local set-constraint. Let B_C be the set of C-bad strings in S and L_C be the set of C-longing strings in S. Note that $L_C \cap B_C = \emptyset$. The set-unsat-value (which we sometimes refer to as the *frustration*) of a set-constraint C with respect to S, is defined by

$$\mathsf{set\text{-}unsat}(C, S) = \frac{|B_C|}{|S|} + \frac{|L_C|}{|S|} \tag{1}$$

Given a collection of m k-local set-constraints $\mathcal{C} = (C_1, ..., C_m)$, its set-unsat-value (or frustration) with respect to S is defined as

$$\mathsf{set\text{-}unsat}(\mathcal{C}, S) = \frac{1}{m} \sum_{i=1}^{m} \mathsf{set\text{-}unsat}(C_i, S). \tag{2}$$

We also define

$$\mathsf{set\text{-}unsat}(\mathcal{C}) = \min_{\substack{S \subseteq \{0,1\}^n \\ S \neq \emptyset}} \{\mathsf{set\text{-}unsat}(\mathcal{C}, S)\}. \tag{3}$$

We say that $\mathcal{C}$ is satisfiable if $\mathsf{set\text{-}unsat}(\mathcal{C}) = 0$ and for $\varepsilon > 0$, we say that $\mathcal{C}$ is ε-frustrated if $\mathsf{set\text{-}unsat}(\mathcal{C}) \geq \varepsilon$.

Notice that the normalization factors in Equation (1) guarantees that the set-unsat-value lies between 0 and 1.[7]

[7] The lower bound is trivial since the value cannot be negative. For the upper-bound, notice that $B_C, L_C \subseteq S$ and $B_C \cap L_C = \emptyset$. This is because bad strings have no neighbors whereas longing strings do. Hence $\frac{|B_C|}{|S|} + \frac{|L_C|}{|S|} \leq \frac{|S|}{|S|} = 1$.

3.2 Intuition and standard CSP as special case

We can present a standard k-CSP instance consisting of constraints $C_1, C_2, ..., C_m$ as an instance for k-SetCSP in the following way: For each constraint $C_\ell, \ell \in \{1, ..., m\}$ of the m constraints in the k-CSP instance, we consider the following set-constraint C'_ℓ: for every k-bit string s that *satisfies* C_ℓ, add the subset $Y_s = \{s\}$ to C'_ℓ. We arrive at a collection $\mathcal{C}$ of m set-constraints, where each set-constraint C'_ℓ in $\mathcal{C}$ consists of single-string sets Y_i corresponding to all strings which satisfy C_ℓ.

We claim that the resulting SetCSP instance has $\mathsf{set\text{-}unsat}(\mathcal{C}) = 0$ if and only if the original CSP instance was satisfiable. Note that in our case, there is no notion of C-neighbors (See Definition 3.5), since all Y_i's contain only a single string. Hence, there are no longing strings; By the definition of $\mathsf{set\text{-}unsat}$ (Equation (1)) the $\mathsf{set\text{-}unsat}(\mathcal{C}, S) = 0$ if and only if all strings in S are good for all set-constraints C', namely each of them satisfies all constraints C in the original k-CSP instance; and moreover, in this case, choosing $S' = \{s\}$ for any string $s \in S$, will give $\mathsf{set\text{-}unsat}(\mathcal{C}, S') = 0$ as well. Hence, there exists a non empty S such that $\mathsf{set\text{-}unsat}(\mathcal{C}, S) = 0$, if and only if there exists a string which satisfies all constraints in the original k-CSP instance.

We give now some intuition about the $\mathsf{set\text{-}unsat}$ quantity (Equation (1)). We first note that it generalizes the by-now-standard notion of (un)satisfiability in CSP, which for a given string, counts the number of unsatisfied constraints, divided by m. We note that in Equation (1), if $S = \{s\}$, then $|B_C|$ is either 0 or 1, depending on whether s satisfies the constraint or not. As previously remarked, in CSP the notion of neighboring and longing strings does not exist, and so L_C will always be empty. Thus, in the case where $S = \{s\}$ and all set-constraints containing single strings, Equation (2) is indeed the number of violated constraints by the string, divided by m - which is exactly the standard CSP unsat used in PCP contexts [Din07]. (In the case of S containing more than one string, but all Y_is are still single-strings, Equation (2) will just be the (non-interesting) average of the unsat of all strings in S).

The interesting case is the general SetCSP case, when the Y_i's contain more than a single string, i.e., when the notions of neighbors and longing strings become meaningful. In this case, the left term in the RHS of Equation (1) quantifies how far the set S is from the situation in which it contains only "good" (not "bad") strings (this can be viewed as the standard requirement) but it adds to it the right term, which quantifies how far S is from being closed to the action of adding neighbors with respect to the set-constraints[8]. Loosely put, the generalization from constraints to set-constraints imposes strong "dependencies" between different strings, and the number of longing strings, L_C, counts to what extent these dependencies are violated.

4 MA-completeness of $\mathsf{SetCSP}_{1/\mathrm{poly(n)}}$

In this section, we show that there is an inverse polynomial ε such that $\mathsf{SetCSP}_\varepsilon$ is MA-complete. We show in Section 4.1 that for all polynomials p and constant k, k-$\mathsf{SetCSP}_{\frac{1}{p}}$ is contained in MA. Then we show in Section 4.2 that there exists some polynomial p such that $6-\mathsf{SetCSP}_{\frac{1}{p(n)}}$ is MA-hard.

[8]Notice that we could have also chosen to define "far" here, by counting the number of strings *outside* of S that are neighbors of S. But since the degree of each string in the graph is bounded, the exact choice of the definition does not really matter; we chose the one presented here since it seems most natural.

4.1 Inclusion in MA

We show in this section that k-$\mathsf{SetCSP}_{\frac{1}{p}}$ is in MA for every constant k and polynomial p. For that, given an instance $\mathcal{C}$ of k-$\mathsf{SetCSP}_{\frac{1}{p}}$, we consider the (undirected) graph $G_{\mathcal{C}} = (\{0,1\}^n, E)$, where $\{x,y\} \in E$ if there exists a set-constraint $C \in \mathcal{C}$ such that x and y are C-neighbors.

Lemma 4.1 (Satisfiable SetCSP instances imply good paths)**.** *If there exists a non-empty $S \subseteq \{0,1\}^n$ such that $\mathsf{set\text{-}unsat}(S,\mathcal{C}) = 0$, then the connected component of any string $x \in S$ in $G_{\mathcal{C}}$ contains only good strings.*

Proof. We prove that S is the union of connected components of $G_{\mathcal{C}}$. In this case, there are no longing strings. The claim will follow since by $\mathsf{set\text{-}unsat}(S,\mathcal{C}) = 0$, all strings in S are good.

Suppose towards contradiction that there exists a string x in S which is connected to a string y outside of S via an edge in $G_{\mathcal{C}}$; that means that x and y are C-neighbors for some set-constraint C. But this means that x is a C-longing string for S, and this contradicts $\mathsf{set\text{-}unsat}(S,\mathcal{C}) = 0$. □

The above implies that in the case of $\mathsf{set\text{-}unsat}(S,\mathcal{C}) = 0$, any path starting from $x \in S$ (x must be a good string in this case) remains in the set of good strings for ever. This is a very important observation for this note. On the other hand:

Lemma 4.2 (Frustrated SetCSP instances implies bad paths)**.** *If for all S, $\mathsf{set\text{-}unsat}(\mathcal{C},S) \geq \frac{1}{p(n)}$, then there exist polynomials q_1 and q_2 such that for every $x \in \{0,1\}^n$, a $q_1(n)$-step lazy random walk*[9] *starting at x reaches a bad string with probability at least $\frac{1}{q_2(n)}$.*

Proof. Let x be some initial (good) string (notice that we can assume that x is a good string since otherwise the lazy random walk reaches a bad string with probability 1). Let G_x be the connected component of x in $G_{\mathcal{C}}$, V_x be the set of vertices (strings) in the connected component We partition V_x to V_x^g, the good strings in V_x (we denote $A = V_x^g$) and $B = V_x \setminus V_x^g$, the bad strings in V_x (a string is bad if it is not good).

We now make a few claims about the parameters of this lazy random walk starting at x. First, notice that each string has at most $m2^k$ neighbors in $G_{\mathcal{C}}$. Secondly, we want to upper bound the size of the edge boundary of S, defined as $\partial_G(S) = \{\{u,v\} \in E : u \in S, v \notin S\}$. We claim that by the conditions of the lemma, it must be that for any $A' \subseteq A$, we have the following bound:

$$|\partial_G(A')| \geq \frac{|A'|}{p(n)}. \tag{4}$$

Assume towards contradiction that $|\partial_G(A')| < \frac{|A'|}{p(n)}$. We have that

$$\mathsf{set\text{-}unsat}(\mathcal{C},A') = \frac{1}{m}\sum_{i=1}^{m} \mathsf{set\text{-}unsat}(C_i,A') = \frac{1}{m}\sum_{i=1}^{m}\frac{|L_{C_i}|}{|A'|} \leq \frac{m\partial_G(A')}{m|A'|} < \frac{1}{p(n)}, \tag{5}$$

where $L_{C_i} \subseteq A'$ is the set of C_i-longing strings in A'. In the second equality we use the fact that there are no bad strings in A', and in the first inequality we use $|L_{C_i}| \leq |\partial_G(A')|$. This follows from the fact that each vertex (string) in L_{C_i} corresponds to a *different* edge in $\partial_G(A')$. We arrive at Equation (5) which is a contradiction to our assumption in the Lemma.

[9] As is defined in Appendix A, in a lazy random walk, at every step we stay in the current vertex with probability $\frac{1}{2}$, and with probability $\frac{1}{2}$ we choose a random neighbor of the current vertex and move to it.

We can now ask how fast does a lazy random walk starting from x, reach an element in B. This is a well known question from random walk theory, and it can be stated as follows.

Lemma 4.3 (Escaping time of high conductance subset)**.** *Let $G = (V = A \cup B, E)$ be a simple (no multiple edges) undirected connected graph, such that for every $v \in A$ $d_G(v) \leq d$, and such that for some $\delta < \frac{1}{2}$, for all $A' \subseteq A$, we have that $|\partial_G(A')| \geq \delta |A'|$. Then a $\left(\frac{16d^2}{\delta^2} \ln \frac{2d|V|}{\delta}\right)$-step lazy random walk starting in any $v \in A$ reaches some vertex $u \in B$ with probability at least $\frac{\delta}{4d}$.*

We defer the proof of this lemma to Appendix A. We apply this lemma for $G = G_x$, $A = V_x^g$, $B = V_x \setminus V_x^g$, $d = m2^k$, and $\delta \geq \frac{1}{p(n)}$, it follows that a $\left(16m^2 2^{2k} p(n)^2 \ln\left(m2^{n+k+1}p(n)\right)\right)$-step lazy random walk starting on any $x \in G_x$ reaches a bad string with probability at least $\frac{1}{4m2^k p(n)}$. □

From the two previous lemmas, we can easily achieve the following.

Corollary 4.4. *For any constant k and polynomial p, k-$\mathsf{SetCSP}_{\frac{1}{p}}$ is in* MA*.*

Proof. The witness for the k-SetCSP instance consists of some string x. Define q_1 and q_2 as the same polynomials of Lemma 4.2. The MA-verification algorithm consists of repeating $nq_2(n)$ times the following process: start from x, perform a $q_1(n)$-step lazy random walk in $G_{\mathcal{C}}$, reject if any of these walks encounters a bad string, otherwise accept.

If $\mathcal{C}$ is a yes-instance, then the prover can provide a string x which belongs to the set S which satisfies the SetCSP instance. From Lemma 4.1, any walk starting from x will never encounter a bad string.

If $\mathcal{C}$ is a no-instance, from Lemma 4.2, each one of the random walks will find a bad string with probability $\frac{1}{q_2(n)}$, and therefore if we perform $nq_2(n)$ lazy random walks, the probability that at least one of them finds a good string is exponentially close to 1. □

4.2 MA-hardness

In this section we show how to reduce any language $\mathsf{L} \in$ MA to a 6-SetCSP instance. More concretely, we provide a reduction from an arbitrary MA-verification algorithm to an instance of 6-SetCSP problem.

Our approach here is to "mimic" the Quantum Cook-Levin theorem due to Kitaev [KSV02], but given that we only need to deal with set of strings and not arbitrary quantum states, our proof can in fact be stated in set-constraints language.

4.2.1 Intuition

In the (classical) Cook-Levin theorem, the verification algorithm for an instance of an NP-language is reduced to a instance of the 3-SAT problem. In this reduction, a different variable is assigned to the value of each location of the tape of the Turing machine at any time step; these variables are used to keep track of the state of the computation at different stages. See Figure Figure 1. Each string, namely an assignment to all these variables, encodes the value of the *entire* history of a single possible computation. The clauses of the Boolean formula check that the history indeed corresponds to a correct propagation of an accepted computation. More precisely, the constraints check if *a*) the input (namely the assignment to the variables associated with the input string in the first time step) is correct; *b*) the assignment of the variables corresponding to any two subsequent time steps is consistent with a correct evolution of each gate in the computation; and c) the output bit, namely the value of the output variable in the *last* time step, is indeed *accept.* If there is a single valid history which ends with accept (namely if x is in the language accepted by the verifier), the resulting k-CSP is satisfiable; the satisfying string encodes the entire *history* of the computation of the verifier. If the k-CSP is satisfiable, it must be a history of an input string accepted by the verifier.

Now, suppose we want to apply a similar construction for some MA verification. The random bits are also given to the verification circuit as an input, and one could hope that the reduction of the Cook-Levin theorem would still work. However, the problem is that there could be *some* choice of random coins that make the verification algorithm to accept even for no-instances, because the soundness parameter is not 0 but some fraction. Hence, the CSP would be satisfiable in this case, even though the input is a NO instance. It is thus not sufficient to verify the existence of a *single* valid history. In order to distinguish between YES and NO instances of the problem, we have to check in the YES case that *all* (or at least many of the) initializations of the random bits lead to accept. The key difficulty is how to check that not only a single string satisfies the constraints, but many.

This is exactly the reason for introducing the *set-constraints.* These set-constraints are able to verify that the random bits are indeed *uniformly random*; then the standard Cook-Levin approach described above can verify that (given the right witness) *each* of them leads to acceptance.

In order to implement such an approach, we first explain how to modify the original Cook-Levin proof, of the NP completeness of satisfiability so that the strings that we check are not entire histories of a verification process of some NP problem; rather, the strings represent *snapshots* of the tape (or evaluations of all bits involved in the circuit at a certain time) for *different time-steps* of the computation. A satisfying assignment is no longer a single string but a whole collection of strings, denoted S, which would be the *collection of all snapshots of a single valid computation, at different times.*[10]

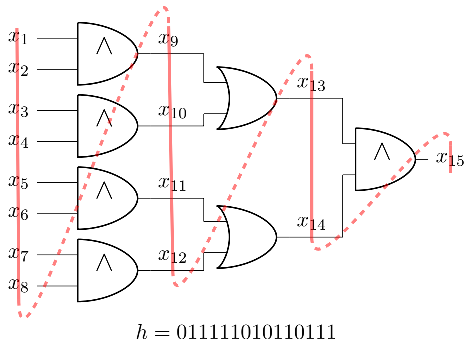
In this case (note that we have still not included random bits in the discussion) we need to show how to create *set-constraints* that verify that the set S really does contain *all* snapshots, and it also needs to verify that S contains nothing else. More precisely, we need to verify that a) the strings in S are consistent with being snapshots of a valid evolution of the computation b) the input is correct in the string corresponding to the first snapshot, and the output is accept in the string corresponding to the last snapshot, and c) the set S indeed contains the whole *history* of the computation, i.e. all the snapshots in a correct evolution, and without any missing step.

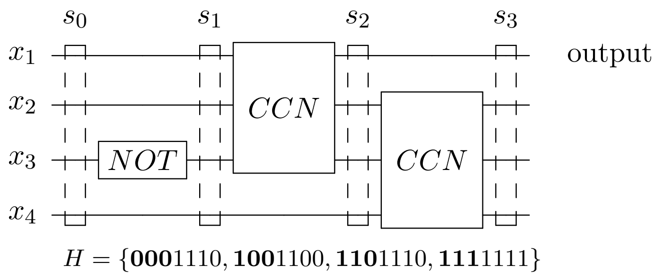
It turns out that this can be done using set-constraints; We depict the differences between the "original" Cook-Levin proof and the "set-constraints" one in Figure 1.

In reality, we need to do this not just for a single evolution but for the evolution over all possible assignments to the random strings. Moreover, we need to enforce that the random bits are indeed random; this requires further set-constraints (technically, this is done below in Equation (6)). This corresponds to the additional step d) which verifies that all evolutions - for all possible random coins - are indeed contained in S.

We note that the essence of the idea appears already when there are no random bits involved at all; the reader is recommended to pretend that no random bits are used, in first reading.

[10] For quantum readers, we note that this reflects the main step in Kitaev's modification of the Cook-Levin theorem, which enabled him to test that entangled quantum states evolved correctly.





s_0
s_1
s_2
s_3
x_1
x_2
x_3
x_4
output
NOT
CCN
CCN
$H = \{\mathbf{000}1110, \mathbf{100}1100, \mathbf{110}1110, \mathbf{111}1111\}$

(a) In the Cook-Levin theorem, a fresh new variable is assigned to every wire of the circuit, and the evolution of the computation can be described as an assignment to the variable such that their values are consistent according to the circuit. In this example, we see a (very simple) circuit, and then the history of the computation on input 01111101. The CSP instance derived from the Cook-Levin theorem ensures that the assignment is indeed the evolution of the circuit for some input (and of course, that the circuit accepts at the end).

(b) In this work, we consider *reversible* circuits and the history of the computation is described by a *set of strings*. We define then the *history-set* of the computation that contains the *snapshot* of every stage of the computation. Each such a string is augmented with a prefix (marked in bold) identifying the timestep, in unary, of the snapshot. These bits, called the "clock", are necessary in order to check the evolution. In this example, we see a very simple circuit[a] that has no auxiliary nor random bits. We show the history-set of the computation on input 1110 and the prefix indicating the number of the timestep (here, 0 to 3), counted in unary. The SetCSP instance derived from the our result ensures that this set contains the snapshots for *all* timesteps of the computation.

[a]The gate CCN is the $CCNOT$ gate which receives some input (a, b, c) and outputs $(a, b, c \oplus ab)$.

Figure 1: Comparison between the evolution of a circuit seen in the Cook-Levin proof and in our work.

We next explain the reduction from MA-verification algorithms to 6-SetCSP in Section 4.2.2 and then we prove its correctness in Section 4.2.3.

4.2.2 The reduction

We assume, without loss of generality, that the MA verification algorithm is reversible, as described in Section 2.3: Given an input $x \in \{0,1\}^n$, we assume a verification circuit C_x whose input is $y \,||\, 0^{a(n)} \,||\, r$, where $y \in \{0,1\}^{p(n)}$ is the polynomial-size MA witness, the middle register consists of the circuit auxiliary bits (needed for reversibility) and $r \in \{0,1\}^{q(n)}$ are the polynomially many random bits used during the MA verification. The circuit C_x consists of T gates $G_1, ..., G_T$, where $G_i \in \{\text{NOT}, \text{CNOT}, \text{CCNOT}\}$.[11] At the end of the circuit, we assume WLOG that the first bit as the output bit.

We describe now the reduction from the MA verification algorithm (compiled in the form described above) into a SetCSP instance $\mathcal{C}$. We will show in the next section that if there is an MA witness that makes the verification algorithm accept with probability 1, then $\mathcal{C}$ is satisfiable, whereas if every witness makes the verifier reject with probability at least $\frac{1}{2}$, then $\mathcal{C}$ is at least inverse polynomially frustrated.

We start with an MA verification circuit C_x (assumed to be reversible, as described in Section 2.3) acting on an input consisting of $a(n)$ 0-bits, witness y of $p(n)$ bits and $q(n)$ random bits. Thus, the number of bits which the reversible verification circuit acts on is $w(n) = a(n)+p(n)+q(n)$. The number of gates is $T(n)$; denote these gates by $G_1, ..., G_T$. Our set-constraint instance $\mathcal{C}$ will act on strings of $s(n) = T(n) + w(n)$ many bits. We omit n from such functions from now on, since it will be clear from the context.

We call the T first bits of such strings the *clock* register and the last w bits the *work* register. The work register comprises of three sub-registers: the witness register (first $p(n)$ bits), the auxiliary register (middle $a(n)$ bits) and the randomness register (last $q(n)$ bits). We want to create set-constraints which force all the strings to be of the form $z \in \{0,1\}^s$ such that $z|_{[T]} = \mathsf{unary}(t)$ for some $t \in [T+1]$ (where $\mathsf{unary}(t)$ denotes the integer t written in unary representation), and $z|_{[s]\setminus[T]}$ represents the *snapshot* of the computation at time t (as explained in Section 4.2.1 above) for the initial string $y \,||\, 0^{a(n)} \,||\, r$, for some witness y and some choice of random bits r.

We will construct $\mathcal{C}$ in such a way that $S \subseteq \{0,1\}^s$ satisfies $\mathcal{C}$ if and only if *i*) it contains *all* snapshots of the computation for some witness y and for *all* bit strings r input to the randomness register; and *ii*) the computations whose snapshots are contained in S are not only correct (meaning also that the auxiliary bits are all initialized to 0) but that they are *accepted* computations (meaning that the output is 1).

We do this by providing set-constraints of four types, as follows.

Clock consistency. We first impose that if $z \in S$, then $z|_{[T]} = \mathsf{unary}(t)$, for some $t \in [T+1]$.

Notice that a string is a valid unary encoding iff it does not contain 01 as a substring. To guarantee that the clock bits are consistent with some unary representation of an allowed t, we add for every $t \in [T]$ the set-constraint C_t^{clock} defined by:

$$Y(C_t^{clock}) = (\{\{00\}, \{10\}, \{11\}\}) \text{ and } J(C_t^{clock}) = (t, t+1).$$

<u>Example:</u> The string $010^{T-2} \,||\, z$ is a bad string for C_1^{clock} since $w|_{J(C_1^{clock})} = 01 \notin Y(C_1^{clock})$.

[11] Recall that the gate CNOT on input a, b outputs $a, b \oplus a$, and the gate CCNOT on input bits a, b, c, outputs $a, b, c \oplus ab$.

Initialization of Input bits and Random bits. Here, we want to check that $0^T \,||\, y \,||\, z \,||\, r$ is not in S whenever $z \neq 0^a$, which enforces the ancillary bits to be initialized to 0. In addition, we want to check that for any witness y, if one string of the form $0^T \,||\, y \,||\, 0^a \,||\, r$ for some r is in S, then for all $r' \in \{0,1\}^q$, we have $0^T \,||\, y \,||\, 0^a \,||\, r'$ in S. In conclusion, we need to check two things: that all auxiliary bits are initialized to 0 and that all possible initializations of the random bits are present.

For each *auxiliary bit* $j \in [a]$ qubit we add a set-constraint C_j^{aux} and for every random bit $j \in [q]$ we add the set constraint C_j^{rand} as follows.

We define

$$Y(C_j^{aux}) = \{\{00\}, \{10\}, \{11\}\} \text{ and } J(C_j^{aux}) = (0, T+p+j),$$

which forces that for $t = 0$ (notice that the unique value of t for which $\mathsf{unary}(t)$ has the first bit 0 is $t = 0$), the j-th auxiliary bit must be 0, because the string (01) is forbidden. For $t \neq 0$ this is not enforced by allowing any value of the j-th auxiliary bit when the first clock bit is 1 (and therefore $t \neq 0$).

Example: The string $0^T \,||\, y \,||\, 10^{a-1} \,||\, r \in S$ is bad for C_0^{aux}.

For the *random bits*, we want to make sure that the j-th random bit has both values 0 and 1, over all the random bits. Therefore we define the constraints C_j^{rand} by

$$Y(C_j^{rand}) = \{\{00, 01\}, \{10\}, \{11\}\} \text{ and } J(C_j^{rand}) = (0, T+p+a+j). \tag{6}$$

These constraints will be useful in the following way. First, the propagation set-constraints that we will define soon, constrain S to make sure that there *exists* a string $0^T \,||\, y \,||\, 0^a \,||\, r$ representing a snapshot at time 0 with some value of the random bits, r, which is indeed in S. The C^{rand} constraints enforce that given the existence of such a string in S, then for any other assignment to the random bits, r', the string $0^T \,||\, y \,||\, 0^a \,||\, r' \in S$. Then, if many of these other strings are not in S, the frustration will be high.

Example: If $s_1 = 0^T \,||\, y \,||\, 0^a \,||\, 0^r \in S$ but $s_2 = 0^T \,||\, y \,||\, 0^a \,||\, 10^{r-1} \notin S$, then s_1 is a C_1^{rand}-longing string in S since s_1 and s_2 are C_1^{rand}-neighbors.

Propagation. Here we want to check that if $\mathsf{unary}(t-1) \,||\, z \in S$ for some $0 < t < T$, then $\mathsf{unary}(t) \,||\, G_t(z) \in S$.

Let us consider the propagation constraint associated with the t-th timestamp (the one corresponding to the application of the tth gate, G_t), for $1 < t < T$. Let us assume that the t-th gate acts on bits $b_{t,1}, ..., b_{t,k}$. For this we add the set-constraint C_t^{prop} defined as follows:

$$Y(C_t^{prop}) = \bigcup_{z \in \{0,1\}^k} \{\{100 \,||\, z, 110 \,||\, G_t(z)\}\} \text{ and } J(C_t^{prop}) = (t-1, t, t+1, b_{t,1}, b_{t,2}, ..., b_{t,k}).$$

For $t = 1$ we simply erase the left clock bit from the above specification:

$$Y(C_1^{prop}) = \bigcup_{z \in \{0,1\}^k} \{\{00 \,||\, z, 10 \,||\, G_1(z)\}\} \text{ and } J(C_1^{prop}) = (1, 2, b_{1,1}, b_{1,2}, ..., b_{1,k}).$$

Likewise if $t = T$ erase the right most clock bit from the above.

Example: If $s_1 = \mathsf{unary}(t) \,||\, z \in S$ but $s_2 = \mathsf{unary}(t+1) \,||\, G_{t+1}(z) \notin S$, s_1 is a C_{t+1}^{prop}-longing string in S, since the two strings are neighbors.

Output. Finally, we need to check that for all strings of the form $1^T \,||\, z$, the first bit of z is 1, namely, the last snapshot corresponds to accept. We define C^{out} such that

$$Y(C^{out}) = \{\{00\},\{01\},\{11\}\} \text{ and } J(C^{out}) = (T, T+1).$$

Here we use the fact that the T-th bit of $\mathsf{unary}(t)$ is 1 iff $t = T$. In this case, if this value is 1, we require that the output bit is 1, otherwise it could have any value.
Example: The string $1^T \,||\, 0 \,||\, z$ is bad for C^{out}.

4.2.3 Correctness

Given some MA-verification circuit C_x, we consider the following 6-SetCSP instance

$$\mathcal{C}_x = (C_1^{clock}, ..., C_T^{clock}, C_1^{aux}, ..., C_a^{aux}, C_1^{rand}, ...C_q^{rand}, C_1^{prop}, ..., C_T^{prop}, C^{out}\}.$$

Let m be the number of terms in $\mathcal{C}_x$ and when it is more convenient to us, we will refer to the set-constraints in $\mathcal{C}_x$ as C_i for $i \in [m]$, where the terms have an arbitrary order. We show now that $\mathcal{C}_x$ is satisfiable if x is a positive instance, and if x is a negative instance, then $\mathcal{C}$ is at least $\frac{1}{10(T+1)qm}$-frustrated (where we remember that q is the number of random coins used by C_x). Notice that m, the number of constraints in $\mathcal{C}_x$, is polynomial in T, which is also polynomial in $|x|$.

We start by proving completeness.

Lemma 4.5 (Yes-instances lead to satisfiable SetCSP instances)**.** *If $x \in \mathsf{L}$, then $\mathcal{C}_x$ is satisfiable.*

Proof. Let y be the witness that makes C_x accept with probability 1, and let

$$S = \{\mathsf{unary}(t) \,||\, G_t...G_1(y, 0^a, r) : r \in \{0,1\}^q, t \in [T+1]\}.$$

By construction, the initialization, clock and propagation constraints are satisfied by S. By the assumption that the MA verification circuit accepts with probability 1, the output constraints are also satisfied by S. □

Next we prove soundness.

Lemma 4.6 (NO-instances lead to frustrated SetCSP instances)**.** *If $x \notin \mathsf{L}$, then $\mathsf{set\text{-}unsat}(\mathcal{C}_x, S) \geq \frac{1}{10(T+1)qm}$ for every non-empty $S \subseteq \{0,1\}^n$.*

Proof. Let $S \subseteq \{0,1\}^n$ be a non-empty set, B the set of bad strings in S (namely a string in S which is C-bad for at least one set-constraint C) and L the set of longing strings in S (namely the strings in S which are C-longing for at least one set-constraint C). Our goal here is to consider a partition $\{K_i\}$ of S, such that for every K_i, $|K_i \cap (B \cup L)| \geq \frac{|K_i|}{10(T+1)q}$, and from this we will show that $\mathsf{set\text{-}unsat}(\mathcal{C}_x, S) \geq \frac{1}{10(T+1)qm}$.

Let us start by defining $S_{ic} \subseteq S$ to be the subset of S with **i**nvalid **c**lock register. Notice that every $x \in S_{ic}$ is bad for at least one clock constraint and therefore $|S_{ic} \cap B| = |S_{ic}|$.

Now we notice that all other strings correspond to some valid clock register whose value (when read as a unary representation of some integer) is in $[T+1]$. Let us partition the strings in $S \setminus S_{ic}$ into disjoint sets H_1,...,H_ℓ (which indicate different history-sets to which the strings belong) as follows. We define the *initial configuration* of some string in $S \setminus S_{ic}$ like this. The string must be of the form $\mathsf{unary}(t) \,||\, z$ for some t and z. Then $\mathsf{initial}(\mathsf{unary}(t) \,||\, z) = \mathsf{unary}(0) \,||\, G_1^{-1}...G_t^{-1}(z)$, which is the assignment of the initial bits that leads to the configuration z at the t'th step. We say then that two strings $s_1, s_2 \in H_i$ iff $\mathsf{initial}(s_1) = \mathsf{initial}(s_2)$ and we abuse the notation and call $\mathsf{initial}(H_i) = \mathsf{initial}(s_1)$. Notice that for $i \neq j$, H_i and H_j are disjoint because the computation is reversible; thus the H_i's constitute a partition of $S \setminus S_{ic}$. Notice also that each H_i contains at most $T+1$ strings, and the different strings in each H_i have different values of the clock register. We call these H_i *history-sets* for the reason that they correspond to a correct propagation of the computation of the circuit C_x for some initialization of all its bits.[12]

Let us first consider the history-sets whose initial configuration is not valid, i.e., it contains **i**nvalid (or non-zero) **a**uxiliary bits: $\mathbf{H}^{ia} = \{H_i : \mathsf{initial}(H_i) = 0^T \,||\, y \,||\, z \,||\, r \text{ for some } z \neq 0^a\}$. We note that for any $H_i \in \mathbf{H}^{ia}$, $\mathsf{initial}(H_i)$ is a C_j^{aux}-bad string in H_i for some j. If this string is in H_i, then we indeed have $|H_i \cap (B \cup L)| \geq |H_i \cap B| \geq 1 \geq \frac{|H_i|}{T+1}$. However, if H_i does not contain its initial string, then consider the minimal t such that $\mathsf{unary}(t) \,||\, z \in H_i$ for some z, and by assumption we have $t > 0$. This means that the string $\mathsf{unary}(t) \,||\, z$ is a C_t^{prop}-longing string, because it is a neighbor of $\mathsf{unary}(t-1) \,||\, G_t^{-1}(z) \notin S$. Hence, for such H_i we have $|H_i \cap (B \cup L)| \geq |H_i \cap L| \geq 1 \geq \frac{|H_i|}{T+1}$. This completes handling all the strings in S within a history set H_i with invalid auxiliary bits in $\mathsf{initial}(H_i)$.

We now need to consider history sets in $\{H_i\} \setminus \mathbf{H}^{ia}$, namely, the history sets whose initial string is of the form $0^T \,||\, y \,||\, 0^a \,||\, r$ for some value of y and r. Let us group these history sets according to y, the value of the witness register. In other words, let us consider the sets of history sets: $\mathbf{H}_y = \{H_i : H_i\text{'s initial string is of the form } 0^T \,||\, y \,||\, 0^a \,||\, r\}$. We fix now some y and the following arguments hold for each y separately. Notice that each $H_i \in \mathbf{H}_y$ corresponds to a computation corresponding to a different initial random string for the witness y. Let us denote the union of strings in all sets in $\mathbf{H}_y$ by $\mathbf{S}_y = \bigcup_{H_i \in \mathbf{H}_y} \{s | s \in H_i\}$. To finish handling all strings, we need to provide a bound on $|\mathbf{S}_y \cap (L \cup B)|$ for all witnesses y.

To proceed, we need some additional notation. We note that $\mathbf{H}_y$ can be written as the union of the history sets which contain their initial string, denoted $\mathbf{H}_y^{start}$, and the rest, denoted $\mathbf{H}_y^{nostart}$. We proceed by considering two cases separately:

1. Let us first consider the simpler case in which $|\mathbf{H}_y^{nostart}| > \frac{|\mathbf{H}_y|}{10}$. As above, we have that each $H_i \in \mathbf{H}_y^{nostart}$ has a longing string, and therefore we have that $|\mathbf{S}_y \cap (L \cup B)| \geq |\mathbf{S}_y \cap L| \geq |\mathbf{H}_y^{nostart}| > \frac{|\mathbf{H}_y|}{10} \geq \frac{|\mathbf{S}_y|}{10(T+1)}$. The last inequality is due to the fact that each $H_i \in \mathbf{H}_y$ contains at most $T+1$ strings. This finishes the treatment of all strings in S, in the case $|\mathbf{H}_y^{nostart}| > \frac{|\mathbf{H}_y|}{10}$.

2. In the second case $|\mathbf{H}_y^{start}| \geq \frac{9|\mathbf{H}_y|}{10}$. We further denote $\mathbf{S}_y^{init} = \{s | s = initial(H_i), H_i \in \mathbf{H}_y^{start}\}$ as the set of the initial strings of each $H_i \in \mathbf{H}_y^{start}$. Again (and for the last time) there are two cases.

   (a) First, let us consider the case when $|\mathbf{S}_y^{init}| \geq 2^{q-1}$. This means that for this fixed y, for most values r of the random bits, the initial string $0^T \,||\, y \,||\, 0^a \,||\, r$ of the history set corresponding to this y and r is present in S_y. We use the facts that $x \notin \mathsf{L}$, and that at least 2/3 of the history sets must lead to rejection by Definition 2.2. From these

[12] More precisely, H_i is just a subset of the correct propagation because it involves only the snapshots in S.

observations, we will conclude that there will be either many bad strings due to the final accept constraint C^{out}, or many longing strings.

Let $\mathbf{Acc}_y = \{H_i \in \mathbf{H}_y : 1^T \,||\, z \in H_i \text{ and } z = 1 \,||\, z'\}$ be the set of history sets in $\mathbf{H}_y$ that accept in the last step. We have that $|\mathbf{Acc}_y|$ is at most the number of $r \in \{0,1\}^q$ which leads the circuit C_x to accept the witness y; since $x \notin \mathsf{L}$, we have that the probability to accept for any y is most $1/3$. Hence $|\mathbf{Acc}_y| \leq \frac{2^q}{3} \leq 2\frac{|\mathbf{H}_y|}{3} \leq \frac{20|\mathbf{H}_y^{start}|}{27}$, where we used the fact that $2^{q-1} \leq |\mathbf{S}_y^{init}| = |\mathbf{H}_y^{start}| \leq |\mathbf{H}_y|$, and in the last inequality, the fact that we are in the case $|\mathbf{H}_y^{start}| \geq 9|\mathbf{H}_y|/10$. Hence, there are at least $\frac{2^q}{10}$ history sets in $\mathbf{H}_y^{start}$ which do not end in accept. Such H_i either contains the string $1^T \,||\, 0 \,||\, z$ (which is bad for C^{out}), or does not contain a final state at all, namely does not contain a state of the form $1^T \,||\, z$ for some z, resulting in a longing string. Thus, there are at least $\frac{2^q}{10}$ strings in $|\mathbf{S}_y \cap (B \cup L)|$; and since $|\mathbf{S}_y| \leq (T+1)|\mathbf{H}_y|$ and $|\mathbf{H}_y| \leq 2^q$, resulting in $|\mathbf{S}_y| \leq 2^q(T+1)$, we have that $|\mathbf{S}_y \cap (B \cup L)| \geq \frac{2^q}{10} \geq \frac{|\mathbf{S}_y|}{10(T+1)}$.

(b) Finally, let us consider the case where $|\mathbf{S}_y^{init}| \leq 2^{q-1}$. This is where we will need to apply conductance arguments. Let G_y^0 be the subgraph of $G_{\mathcal{C}}$[13] induced by the vertices $R_y = \{0^T \,||\, y \,||\, 0^a \,||\, r : r \in \{0,1\}^q\}$. Notice that G_y^0 is isomorphic to the q-dimensional hypercube. This is true because the only remaining edges on G_y^0 come from the set-constraints C_j^{rand}. Notice also that $\mathbf{S}_y^{init}$ is a subset of the vertices of G_y^0. We now apply Lemma A.2 which states that the conductance of the q-dimensional hypercube is $\frac{1}{q}$. Applying this lemma to the graph G_y^0 and the subset of its vertices, $\mathbf{S}_y^{init}$ we conclude that $\frac{|\partial_{G_y^0}(\mathbf{S}_y^{init})|}{q|\mathbf{S}_y^{init}|} \geq \frac{1}{q}$, where we have used the fact that all vertices in G_y^0 have the same degree, q, and the fact that we are now considering the case $|\mathbf{S}_y^{init}| \leq 2^{q-1}$. We can conclude then that there exists at least $|\mathbf{S}_y^{init}|$ edges in the cut $(\mathbf{S}_y^{init}, R_y \setminus \mathbf{S}_y^{init})$. Since each vertex in G_y^0 (in particular, each vertex in $\mathbf{S}_y^{init}$) has q neighbors, it means that there exists at least $\frac{|\mathbf{S}_y^{init}|}{q}$ longing strings in $\mathbf{S}_y^{init}$. We conclude this case by noticing that

$$|\mathbf{S}_y \cap (L \cup B)| \geq |\mathbf{S}_y \cap L| \geq \frac{|\mathbf{S}_y^{init}|}{q} = \frac{|\mathbf{H}_y^{start}|}{q} \geq \frac{9|\mathbf{H}_y|}{10q} \geq \frac{9|\mathbf{S}_y|}{10q(T+1)},$$

where in the second inequality we use the fact that $\mathbf{S}_y^{init} \subseteq \mathbf{S}_y$ and there are at least $\frac{|\mathbf{S}_y^{init}|}{q}$ longing strings in $\mathbf{S}_y^{init}$, the equality follows since $|\mathbf{H}_y^{start}| = |\mathbf{S}_y^{init}|$, the third inequality follows from our assumption that $|\mathbf{H}_y^{start}| \geq \frac{9|\mathbf{H}_y|}{10}$ and finally we have that $|H_i| \leq T+1$ (and therefore $|\mathbf{H}_y| \geq \frac{|\mathbf{S}_y|}{T+1}$).

To finish the proof, notice that $S = S_{ic} \cup \bigcup_{H \in \mathbf{H}^{ia}} H \cup \bigcup_y \mathbf{S}_y$. Since each of these subsets has at least a $\frac{1}{10(T+1)q}$-fraction of bad strings or longing strings, we have that $|B \cup L| \geq \frac{|S|}{10(T+1)q}$. It follows that

$$\mathsf{set\text{-}unsat}(\mathcal{C}_x, S) = \frac{1}{m}\sum_i \left(\frac{|B_{C_i}|}{|S|} + \frac{|L_{C_i}|}{|S|}\right) \geq \frac{|B|}{m|S|} + \frac{|L|}{m|S|} \geq \frac{|B \cup L|}{m|S|} \geq \frac{1}{10(T+1)qm},$$

finishing the proof. $\square$

[13]Here, we use the same notation as Section 4.1.

5 Acknowledgements

We are grateful to Umesh Vazirani for asking the right question that led to this note. D.A. is grateful for the support of ISF grant 1721/17. Part of this work was done while A.G. was visiting the Simons Institute for the Theory of Computing.

References

[AG19] Dorit Aharonov and Alex B. Grilo. Stoquastic PCP vs. Randomness. In *FOCS 2019*, 2019.

[AvDK$^+$08] Dorit Aharonov, Wim van Dam, Julia Kempe, Zeph Landau, Seth Lloyd, and Oded Regev. Adiabatic quantum computation is equivalent to standard quantum computation. *SIAM Review*, 50(4):755–787, 2008.

[Bab85] László Babai. Trading group theory for randomness. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 421–429, 1985.

[BFNW93] László Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. Bpp has subexponential time simulations unless exptime has publishable proofs. *Comput. Complex.*, 3(4):307–318, October 1993.

[Bra14] Sergey Bravyi. Monte Carlo simulation of stoquastic Hamiltonians. *arXiv preprint arXiv:1402.2295*, 2014.

[BT09] Sergey Bravyi and Barbara M. Terhal. Complexity of stoquastic frustration-free hamiltonians. *SIAM J. Comput.*, 39(4):1462–1485, 2009.

[Din07] Irit Dinur. The PCP Theorem by Gap Amplification. *Journal of the ACM*, 54(3), 2007.

[Dru11] Andrew Drucker. A pcp characterization of am. In *Proceedings of the 38th International Colloquim Conference on Automata, Languages and Programming - Volume Part I*, ICALP'11, 2011.

[GT12] Shayan Oveis Gharan and Luca Trevisan. Approximating the expansion profile and almost optimal local graph clustering. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 187–196, 2012.

[GZ11] Oded Goldreich and David Zuckerman. Another proof that BPP $\subseteq$ PH (and more). In *Studies in Complexity and Cryptography*, pages 40–53. Springer-Verlag, 2011.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4), March 1999.

[IKW02] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: exponential time vs. probabilistic polynomial time. *J. Comput. Syst. Sci.*, 65(4):672–694, 2002.

[IW97] Russell Impagliazzo and Avi Wigderson. P = bpp if e requires exponential circuits: Derandomizing the xor lemma. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 220–229. ACM, 1997.

[JS89] Mark Jerrum and Alistair Sinclair. Approximating the permanent. *SIAM J. Comput.*, 18(6):1149–1178, 1989.

[KI04] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1):1–46, Dec 2004.

[KSV02] Alexei Kitaev, A Shen, and M N Vyalyi. *Classical and quantum computation*. Graduate studies in mathematics. American mathematical society, Providence (R.I.), 2002.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994.

[RV11] Satish Rao and Umesh Vazirani. Lecture 9. cs270, cs270: Algorithms. https://people.eecs.berkeley.edu/~satishr/cs270/sp11/rough-notes/Sparse-cuts-spectra.pdf, 2011. Accessed: 2019-11-14.

[STV01] Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the xor lemma. *Journal of Computer and System Sciences*, 62(2):236 – 266, 2001.

[SW] Peter Shor and Ryan Williams. Mathoverflow: Complete problems for randomized complexity classes. https://mathoverflow.net/questions/34469/complete-problems-for-randomized-complexity-classes. Accessed: 2019-01-14.

[ZF87] Stathis Zachos and Martin Furer. Probabilistic quantifiers vs. distrustful adversaries. In *Seventh Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 443–455, 1987.

A Graph theory and random walks

For some undirected graph $G = (V, E)$, and $v \in V$, let $d_G(v) = |\{u \in V : \{v, u\} \in E\}|$ be the degree of v. We may define a weight function for the edges $w : E \to \mathbb{R}^+$, and if it is not explicitly defined, we assume that $w(\{u, v\}) = 1$ for all $\{u, v\} \in E$. We slightly abuse the notation and for some $E' \subseteq E$, we denote $w(E') = \sum_{e \in E'} w(e)$.

For some set of vertices $S \subseteq V$, we define the boundary of S as $\partial_G(S) = \{\{u, v\} \in E : u \in S, v \notin S\}$, the neighbors of S as $N_G(S) = \{u \in \overline{S} : v \in S, \{u, v\} \in E\}$ (where $N_G(u)$ for some $u \in V$ is shorthand for $N_G(\{u\})$); and finally the volume of S $vol_G(S) = \sum_{u \in S, v \in N_G(u)} w(\{u, v\})$ and the conductance of S, $\phi_G(S) = \frac{w(\partial_G(S))}{vol_G(S)}$.

We define notions related to random walks on weighted graphs; definitions are taken from [GT12]. One step in the random walk on the weighted graph G starting on vertex v is defined by moving to vertex u with probability $p_{v,u} = \begin{cases} \frac{w(\{u,v\})}{vol_G(\{v\})}, & \text{if } \{u, v\} \in E \\ 0, & \text{otherwise} \end{cases}$. One step in the *lazy* random walk on G starting on v is defined by staying in the same vertex with probability $\frac{1}{2}$, or with probability $\frac{1}{2}$, moving according to the above random walk. The conductance of the weighted graph is defined as

$$\phi(G) = \min_{\substack{S \subseteq V \\ vol_G(S) \leq \frac{vol_G(V)}{2}}} \phi_G(S).$$

The (unique) stationary distribution π_G of the lazy random walk on a connected weighted graph G is given by the following probability distribution on the vertices: $\pi_G(v) = \frac{vol_G(\{v\})}{vol_G(V)}$. The following lemma connects the conductance of a graph and its *mixing time*, i.e., the number of steps of a lazy random walk on G starting at an arbitrary vertex such that the distribution on the vertices is close to the stationary distribution.

Lemma A.1 (High conductance implies small mixing time [JS89])**.** *Let $G = (V, E)$ be a undirected connected graph. For any $u, v \in V$, a $\left(\frac{2}{\phi(G)^2/4} \ln \frac{1}{\varepsilon \min\{\pi_G(v)\}}\right)$-step lazy-random walk starting in u ends in v with probability $p^T_{u,v} \in [\pi_G(v) - \varepsilon, \pi_G(v) + \varepsilon]$.* [14]

We now prove Lemma 4.3

Lemma 4.3 (restated) (Escaping time of high conductance subset)**.** Let $G = (V = A \cup B, E)$ be a simple (no multiple edges) undirected connected graph such that for every $v \in A$ $d_G(v) \leq d$, and such that for some $\delta < \frac{1}{2}$, for all $A' \subseteq A$, we have that $|\partial_G(A')| \geq \delta|A'|$. Then a $\left(\frac{16d^2}{\delta^2} \ln \frac{2d|V|}{\delta}\right)$-step lazy random walk starting in any $v \in A$ reaches some vertex $u \in B$ with probability at least $\frac{\delta}{4d}$.

Proof. Let us consider the graph $G' = (V', E')$ where $V' = A \cup \{b^*\}$ and $E' = \{\{u, v\} \in E : u, v \in A\} \cup \{\{u, b^*\} : u \in A, v \in B, \{u, v\} \in E\}$ (as a set, duplicate edges are removed). In other words, G' can be seen as the graph derived when we contract all vertices in B to one vertex (and call this vertex b^*) and contract the parallel edges that appear when we perform the contraction of the vertices into a single edge. For some $u \in A$, we define the weight of the edge $\{u, v\} \in E'$ as

$$w(\{u, v\}) = \begin{cases} 1, & \text{if } v \in A \\ |\{\{u, v'\} : \{u, v'\} \in E, v' \in B\}|, & \text{if } v = b^* \end{cases}.$$

Notice that the probability that a T-step weighted lazy random walk on G' starting in $v \in A$ passes by the vertex b^* is exactly the same as the probability that a T-step lazy random walk on G starting in v passes by some vertex in B. Therefore, we continue by lower bounding the former, and the statement will follow.

Let us start by analyzing $\phi(G')$. Notice that for all $A' \subseteq A$, it follows that

$$\phi_{G'}(A') = \frac{|\partial_{G'}(A')|}{vol_{G'}(A')} = \frac{|\partial_G(A')|}{\sum_{v \in A'} 2d_G(v)} \geq \frac{\delta|A'|}{d|A'|} = \frac{\delta}{d}. \tag{7}$$

[14] We use Corollary 2.2 of [JS89], along with the remark (a) after it. The latter states that the conductance of a graph is divided by 2 when we consider the weighted graph induced by the *lazy* random walk (this induced graph is naturally defined by adding a self loop to each vertex, and giving it the same weight as the sum of the weight of all other edges going out of the node). In Lemma A.1, we use the conductance of the original graph G and account for the fact that we are applying a lazy random walk on it by dividing the conductance by 2. Importantly, the stationary distribution remains the same.

Let $S = \{b^*\} \cup A'$ for some $A' \subseteq A$ such that $vol(S) \leq \frac{vol_{G'}(V')}{2}$, we have that

$$\phi_{G'}(S) = \frac{w(\partial_{G'}(S))}{vol_{G'}(S)} \geq \frac{w(\partial_{G'}(\overline{S}))}{vol_{G'}(\overline{S})} = \frac{|\partial_G(\overline{S})|}{\sum_{v \in \overline{S}} d_G(v)} \geq \frac{|\partial_G(\overline{S})|}{d|\overline{S}|} \geq \frac{\delta|\overline{S}|}{d|\overline{S}|} = \frac{\delta}{d}, \tag{8}$$

where in the first inequality we use the fact that $\partial_{G'}(S) = \partial_{G'}(\overline{S})$ and $vol_{G'}(S) \leq vol_{G'}(\overline{S})$, in the second equality we use the definition of w, in the second inequality we use the fact that $\overline{S} \subseteq A$ and every vertex in A has degree at most d, and the third inequality follows since $\overline{S} \subseteq A$, and so we can use the assumption of the lemma. From Equations (7) and (8), we have that $\phi(G') \geq \frac{\delta}{d}$.

We now analyze the stationary distribution of the lazy random walk associated with G'. Since $\delta|A| \leq vol_{G'}(\{b^*\}) \leq d|A|$ and for any $v \in A$, $1 \leq vol_{G'}(\{v\}) \leq d$, we have that $vol_{G'}(V) \leq (\delta + d)|A| \leq 2d|A|$,

$$\pi_{G'}(b^*) = \frac{vol_{G'}(\{b^*\})}{vol_{G'}(V)} \geq \frac{\delta|A|}{2d|A|} = \frac{\delta}{2d},$$

and for any $v \in A$,

$$\pi_{G'}(v) = \frac{vol_{G'}(\{v\})}{vol_{G'}(V)} \geq \frac{1}{2d|A|} \geq \frac{1}{2d|V|}.$$

We now apply Lemma A.1, using $\varepsilon = \frac{\delta^2}{2d}$, $\min\{\pi_{G'}(v)\} \geq \frac{1}{2d|V|}$ and $\phi(G') \geq \frac{\delta}{d}$. We have that for

$$T = \frac{8}{\phi(G')^2} \ln \frac{1}{\varepsilon \min\{\pi_{G'}(v)\}} \leq \frac{8 \cdot d^2}{\delta^2} \ln \frac{4d^2|V|}{\delta^2} \leq \frac{16d^2}{\delta^2} \ln \frac{2d|V|}{\delta},$$

a T-step lazy random walk starting from any vertex end in b^* with probability at least $\frac{\delta - \delta^2}{2d}$, which is a lower bound on the probability that the lazy random walk passes by b^*. Since $\frac{\delta-\delta^2}{2d} \geq \frac{\delta}{4d}$ this finishes the proof. □

A.1 Hypercube

Given some parameter q, the q-dimensional hypercube is the graph where the vertices are all possible q-bit strings and two vertices are connected if the corresponding strings differ on exactly one bit.

We use the following well-known fact about the expansion of the hypercube.

Lemma A.2 ([RV11]). *Let G be the q-dimensional hypercube. Then $\phi(G) = \frac{1}{q}$.*