Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 43 (2020)

Two combinatorial MA-complete problems

Abstract

Despite the interest in the complexity class MA, the randomized analog of NP, there is just a few known natural (promise-)MA-complete problems. The first such problem was found by Bravyi and Terhal (SIAM Journal of Computing 2009); this result was then followed by Crosson, Bacon and Brown (PRE 2010) and then by Bravyi (Quantum Information and Computation 2015). Surprisingly, two of these problems are defined using terminology coming from quantum computation, while the third is inspired by quantum computation and still keeps a physical terminology. This fact makes it hard for *classical* complexity theorists to study these problems, and prevents potential progress, e.g., on the important question of derandomizing MA.

In this note we define two new natural combinatorial problems and we prove their MA-completeness. The first problem, that we call approximately-clean approximate-connected-component (ACAC), gets as input a succinctly described graph, some of whose vertices are marked. The problem is to decide whether there is a connected component whose vertices are *all unmarked*, or the graph is *far* from having this property. The second problem, called SetCSP, generalizes in a novel way the standard constraint satisfaction problem (CSP) into constraints involving *sets* of strings.

Technically, our proof that SetCSP is MA-complete is a fleshing out of an observation made in (Aharonov and Grilo, FOCS 2019), where it was noted that a restricted case of Bravyi and Terhal's MA complete problem (namely, the *uniform* case) is already MA complete; and, moreover, that this restricted case can be stated using classical, combinatorial language. The fact that the first, arguably more natural, problem of ACAC is MA-hard follows quite naturally from this proof as well; while containment of ACAC in MA is simple, based on the theory of random walks.

We notice that this work, along with a translation of the main result of Aharonov and Grilo to the SetCSP problem, implies that finding a gap-amplification procedure for SetCSP (in the spirit of the gap-amplification procedure introduced in Dinur's PCP proof) would imply MA=NP. In fact, the problem of finding gap-amplification for SetCSP is *equivalent* to the MA=NP problem. This provides an alternative new path towards the major problem of derandomizing MA. Deriving a similar statement regarding gap amplification of a natural restriction of ACAC remains an open question.

ISSN 1433-8092

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], however the proof remains illusive. It is even open to show that every problem in MA can be solved 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 that deciding if a given stoquastic k-Local Hamiltonian[2] is frustration-free or at least inverse polynomially frustrated, is promise-MA-complete. Bravyi [Bra14] also proved MA-completeness of yet another quantum Hamiltonian problem. A third MA-complete problem was proposed by Crosson, Bacon and Brown [CBB10], inspired by quantum adiabatic computation.[3]

This leaves us in a situation in which the known MA complete problems are not stated using natural or standard complexity theory terminology. 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][4]. 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 delay progress on them, and make it hard for *classical* complexity theorists to study them.

The goal of this work is to provide classically, combinatorially-defined complete problems for MA. We hope that these definitions lead to further understanding of the class MA and the MA vs. NP question.

One of these problems, SetCSP, is morally based on the MA complete problem of [BT09], while the other problem, ACAC, seems to be a rather natural problem on graphs.

The definition of SetCSP as well as the proof of its MA-completeness rely on a simple but crucial insight: we can translate "testing history states", a notion familiar in quantum complexity theory, into constraints on *sets* of strings. This idea is used here throughout, but in fact can be used to translate also other quantum results related to stoquastic Hamiltonians, to the classical combinatorial language of constraints on *sets of strings* (see Section 4.4.1 as well as Figure 1 for more intuition). However though this translation idea can be viewed as standing at the heart of

[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.

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

[3] Crosson et. al.'s problem asks about the properties of the Gibbs distribution corresponding to the temperature T of a specific family of *classical*, rather than quantum, physical systems, defined using local classical Hamiltonians. Though inspired by adiabatic quantum computation, this problem is in fact defined using *classical* terminology, since the classical Hamiltonians can be viewed as constraint satisfaction systems. Yet, we note that its definition uses a layer of physical notation, involving Gibbs distribution and temperature. Moreover, when stated using classical terminology, the input to this problem is restricted, in a fairly contrived way, to sets of classical constraints which can be associated with a (noisy) deterministic circuit. Both of these aspects seem to make handling this problem using standard combinatorial tools difficult or at least not very natural.

[4] 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.

many of the definitions and results presented here, it is in fact *not* needed to understand it in order to understand the proofs.

Based on this idea, as well as 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, we can prove the MA-hardness proof of the SetCSP problem as a translation of the MA-hardness proof given in [BT09] into a classical language. We notice that the proof of [BT09] on its own heavily relies on the Quantum Cook-Levin proof of Kitaev [KSV02].

It turns out that there is an easy reduction from the problem SetCSP to that of ACAC, and therefore the proof that ACAC is MA-hard follows immediately.

Interestingly, the proof of containment in MA is rather simple for ACAC (it is an easy application of a known result from random walk theory.) Finally, using the same reduction, we arrive at a proof that the SetCSPproblem is also in MA[5].

We stress that we present all proofs here avoiding any quantum notation or quantum jargon what-so-ever. The main contribution of this work is in the definitions themselves, initiated by the small but important conceptual idea of the translation mentioned above; this translation thus provides two new, combinatorial, natural MA-complete problems, which we believe are amenable to research in a language familiar to (classical) computer scientists.

We now describe the problems. In the ACAC problem, we consider an exponentially large graph, accompanied with a function on the vertices that marks some of them. Both graph and the function on the vertices are given implicitly (and succinctly) by a polynomial size circuit. We then ask if there exists a connected component of the graph that is "clean" (meaning that all of its vertices are *unmarked*) or if the graph is ε-far from having this property. The notion of "far" is defined as follows: every set of vertices which is close to being a connected component (i.e. its expansion is smaller than ε) must have at least an ε-fraction of its vertices marked. In other words, either a set is ε-far from a connected component (i.e. has large expansion) or at least ε fraction of its vertices are marked. We call this problem *approximately-clean approximate-connected-component* ($\mathsf{ACAC}_\varepsilon$).

Our second MA-complete problem, called the *Set-Constraint Satisfaction Problem*, or SetCSP, is a somewhat unexpected generalization of the standard Constraint Satisfaction Problem (CSP). While a constraint in CSP acts on a single string (deciding if it is valid or not), the generalized constraints act on *sets* of strings. We call the generalized constraints *set-constraints* (see Section 4 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 set of strings S is ε-far from satisfying this set-constraint collection (see Definitions 4.2 and 4.8 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 show the following three claims: *i*) for every inverse polynomial ε, we have that $\mathsf{ACAC}_\varepsilon$ is in MA (Corollary 3.5); *ii*) there exists an inverse polynomial function $\varepsilon = \varepsilon(n)$ such that $\mathsf{SetCSP}_\varepsilon$ is MA-hard (Lemma 4.9); and *iii*) for all functions $\varepsilon = \varepsilon(n) > 0$, there is a polynomial-time reduction from $\mathsf{SetCSP}_\varepsilon$ to $\mathsf{ACAC}_{\varepsilon/2}$.Together, these facts imply the following results (which are proven in Section 5).

Theorem 1.1. *There exists some polynomial $p(x) = \Theta(1/x^3)$ such that for every inverse polynomial $p' < p$, the problems $\mathsf{SetCSP}_{p'(m)}$ and $\mathsf{ACAC}_{p'(m)/2}$ are* MA*-complete, where m is the size of the* SetCSP *or* ACAC *instance.*

[5] This in fact gives a significantly simpler version than[BBT06] of the proof of containment in MA, in the restricted case of uniform stoquastic Hamiltonians.

We stress that the definitions of both these problems and the proofs presented in this paper only use standard combinatorial concepts from set- and graph-theory.

1.1 Conclusions, future work and open questions

In a similar way to what is done in this note, we claim that it is possible to translate several other results from the stoquastic local Hamiltonian language to the SetCSP language, bringing us to very interesting conclusions.

First, one could strengthen the MA-hardness of SetCSP (Lemma 4.9), whose proof is a translation of the proof of the quantum Cook-Levin theorem [KSV02], to the restricted case in which the constraints are set on a 2D lattice. To do this, one could consider the result of [AvDK$^+$08] where they prove the QMA-hardness of local Hamiltonians on a 2D lattice (considering only the restricted family of stoquastic Hamiltonians), and translate it to the SetCSP language. In this way, it can be proven that $\mathsf{SetCSP}_\varepsilon$ with inverse polynomial ε, and with set-constraints arranged on a $2D$-lattice with constant size alphabet is still MA-hard.[6] We omit here the details of the proof, since it is a straight forward translation of [AvDK$^+$08], similarly to what's done in Lemma 4.9. This leads to the following statement.

Lemma 1.2. *There exists some polynomial p such that for every inverse polynomial $p' < p$, $\mathsf{SetCSP}_{p'}$ is* MA*-hard even when each bit participates in $O(1)$ set-constraints, and each set-constraint acts on $O(1)$ bits.*

We also claim that the main result of [AG19], which states that some problem called stoquastic local Hamiltonian with constant gap is in NP, can be translated to SetCSP language, using again the same translation. This means that the *gapped* version of the SetCSP problem is in NP. By the gapped version of the problem, we mean $\mathsf{SetCSP}_\varepsilon$ when ε is a constant; and where we also require that 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. More concretely, we have the following.

Lemma 1.3. *For any constant ε and constants k and d, $\mathsf{SetCSP}_\varepsilon$ is in* NP *if each bit participates in d set-constraints, and each set-constraint acts on k bits.*

Together these two results lead to a very important and surprising equivalence[7]:

Proposition. MA=NP iff there is a gap amplification reduction[8] for SetCSP. The existence of a gap amplification reduction means that there exists a constant $\varepsilon > 0$, such that for every polynomial p, there is a polynomial time reduction from $\mathsf{SetCSP}_{1/p(n)}$ to $\mathsf{SetCSP}_\varepsilon$.

We note that it is easy to see that MA=NP implies such gap-amplification for SetCSP: if MA = NP, then we can reduce $\mathsf{SetCSP}_{1/poly}$, which is in MA-complete by Theorem 1.1, to $\mathsf{CSP}_{O(1)}$, which is NP-complete by the PCP theorem [Din07]; then, since every CSP instance is also a SetCSP instance with the same parameters, we have the gap-amplification. 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 long standing open problem of derandomizing MA.

[6] We conjecture that this hardness result works even with binary alphabet and we leave such a statement for future work.

[7] This equivalence was highlighted in [AG19] in a quantum language of stoquastic Hamiltonians

[8] In the same sense as Dinur's gap amplification reduction for CSP[Din07]

Finally, we leave as an open problem deriving such a natural statement regarding gap amplification for the ACAC problem. Though the two problems are technically very related, defining a natural restricted version of the gapped ACACproblem, so that the [AG19] result would apply to show containment in NP (similarly to SetCSP with constant ε, locality k and degree d) remains open. The problem is that the locality notions don't have a very natural analogues in the graph language of ACAC.

Organization We provide notation and a few basic notions in Section 2. In Section 3, we define the Approximately-clean approximate-connected-component problem and prove its containment in MA. The definition of SetCSP and the proof of its MA-hardness are in Section 4. The reduction from SetCSP to ACAC appear in Section 5. In Appendices A to C we provide some basic background on computational complexity and random walks, and prove some standard technical results we need. In Appendix D, we also prove the PSPACE-completeness of the exact version of the ACAC problem.

2 Preliminaries

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 by $x \,||\, y$ their concatenation. For a string of bits x, $|x|$ denotes the number of bits in x.

2.1 Complexity classes

A (promise) problem $A = (A_{yes}, A_{no})$ consists of two non-intersecting sets $A_{yes}, A_{no} \subseteq \{0,1\}^*$. For completeness, we add in Appendix A the definitions of the two main complexity classes that are considered in this work: NP and MA.

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].

2.2 Reversible circuits

It is folklore that the verification algorithms for NP and MA can be converted into a uniform family of polynomial-size Boolean circuits, made of reversible gates, $\{\mathrm{NOT}, \mathrm{CNOT}, \mathrm{CCNOT}\}$ by making use of additional auxiliary bits initialized to 0, with only linear overhead. For randomized circuits, we can also assume the circuit uses only reversible gates, by assuming that the random bits are part of input. See Appendix B for more details.

Let $G \in \{\mathrm{NOT}, \mathrm{CNOT}, \mathrm{CCNOT}\}$ be a gate to be applied on the set of bits 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 Approximately-clean approximate-connected-component problem

In this section, our goal is to present the approximately-clean approximate-connected-component problem and prove its containment in MA. But before that, we explain the *exact* version of this problem.

For a fixed parameter n, we consider a graph G of 2^n nodes, which is described by a classical circuit C_G of size (number of gates) poly(n), as follows. For simplicity, we represent each vertex of G as an n-bit string, and C_G, on input $x \in \{0,1\}^n$, outputs the (polynomially-many) neighbors of x in G.[9] We are also given a circuit C_M, which when given input $x \in \{0,1\}^n$, outputs a bit indicating whether the vertex x is marked or not.

We define the clean connected component problem (CCC) which, on input (C_G, C_M), asks if G has a connected component where *all vertices are unmarked* or if all connected components have at least one marked element. We give now the formal definition of this problem.

Definition 3.1 (Clean connected component(CCC))**.** Fix some parameter n. An instance of the clean connected component problem consists of two classical poly(n)-size circuits C_G and C_M. C_G consists of a circuit succinctly representing a graph with $G = (V, E)$ where $V = \{0,1\}^n$ and each vertex has degree at most poly(n). On input $x \in \{0,1\}^n$, C_G outputs all of the neighbors of x. C_M is a circuit that on input $x \in \{0,1\}^n$, outputs a bit. The problem then consists of distinguishing the two following cases:
Yes. There exists one non empty connected component of G such that C_M outputs 0 on all of its vertices.
No. In every connected component of G, there is at least one vertex for which C_M outputs 1.

We show in Appendix D that this problem is PSPACE-complete. Our focus here is to study the *approximate* version of CCC, where we ask whether G has a clean connected component or it is *far* from having this property, meaning that for every set of vertices S, the boundary[10] of S is at least $\varepsilon|S|$ (and therefore S is far from being a connected component), or, if the boundary is small (i.e. S is close to a connected component) it contains at least $\varepsilon|S|$ marked elements. Here is the definition of the problem.

Definition 3.2 (Approximately-clean approximate-connected-component($\mathsf{ACAC}_\varepsilon$))**.** Same as Definition 3.1 with the following difference for no-instances:
No. For all non empty $S \subseteq V$, we have that either

$$|\partial_G(S)| \geq \varepsilon|S| \quad \text{or} \quad |\{x \in S : C_M(x) = 1\}| \geq \varepsilon|S|.$$

In the remainder of this section, we show that $\mathsf{ACAC}_\varepsilon$ is in MA for every inverse-polynomial ε.

3.1 Inclusion in MA

The idea of the proof is as follows. The prover sends a vertex that belongs to the (supposed) clean connected component and then the verifier performs a random-walk on G for sufficiently (but still polynomially-many) steps and rejects if the random walk encounters a marked element.

[9] We notice that usually we succinctly describe graphs by considering a circuit that, on input (x, y), outputs 1 iff x is connected to y. In our result it is crucial that given x, we are able to efficiently compute *all* of its neighbors.

[10] The boundary of a set of vertices $S \subseteq V$ is defined as $\partial_G(S) = \{\{u, v\} \in E : u \in S, v \notin S\}$.

In order to prove that this verification algorithm is correct, we first prove a technical lemma regarding the random-walk on no-instances.

Lemma 3.3 (In NO-instances the random walk reaches marked nodes quickly)**.** *Let ε be an inverse-polynomial function of n. If (C_G, C_M) is a no-instance of $\mathsf{ACAC}_\varepsilon$, 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*[11] *starting at x reaches a marked vertex with probability at least $\frac{1}{q_2(n)}$.*

Proof. Let x be some initial (unmarked) vertex (notice that we can assume that x is unmarked since otherwise the lazy random walk reaches a marked vertex with probability 1). Let G_x be the connected component of x in G and V_x be the set of vertices in the connected component of x. We partition V_x into A, the unmarked vertices in V_x and $B = V_x \setminus A$, the marked vertices in V_x.

We want to upper bound the size of the edge boundary of any set $S \subseteq A$ which contains only unmarked strings. We claim that by the conditions of the lemma, it must be that for any $S \subseteq A$, we have the following bound:

$$|\partial_G(S)| \geq \varepsilon |S|, \tag{1}$$

otherwise S would contradict the fact that (C_G, C_M) is a no-instance, since it only contains unmarked vertices.

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 3.4 (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 C and we apply it using $G = G_x$, A, B, d is the (poly(n)-large) maximum degree of G, and $\delta \geq \varepsilon$. It follows that a $\left(\frac{16d^2}{\varepsilon^2}\left(n + \ln\left(2d/\varepsilon\right)\right)\right)$-step lazy random walk starting on any $x \in G_x$ reaches a marked vertex with probability at least $\frac{\varepsilon}{4d}$. □

From the previous lemma, we can easily achieve the following.

Corollary 3.5. *For any inverse-polynomial ε, $\mathsf{ACAC}_\varepsilon$ is in* MA.

Proof. The witness for the ACAC instance consists of some string x, which is supposed to be in a clean connected component of G. Define q_1 and q_2 as the same polynomials of Lemma 3.3, namely, let $q_1 = \left(\frac{16d^2}{\varepsilon^2}\left(n + \ln\left(2d/\varepsilon\right)\right)\right)$ and $q_2 = \frac{4d}{\varepsilon}$. 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, reject if any of these walks encounters a marked vertex, otherwise accept.

If (C_G, C_M) is a yes-instance, then the prover can provide a vertex x which belongs to the clean connected component. Any walk starting from x remains in its connected component and therefore it will never encounter a marked vertex and the verifier accepts with probability 1.

If (C_G, C_M) is a no-instance, from Lemma 3.3, each one of the random walks will find a marked vertex with probability at least $\frac{1}{q_2(n)}$, and therefore if we perform $nq_2(n)$ lazy random walks, the probability that at least one of them finds a marked vertex is exponentially close to 1. □

[11] As is defined in Appendix C, 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 uniformly at random and move to it.

4 Set-Constraint Satisfaction Problem

In this section we present the Set Constraint Satisfaction Problem (SetCSP), and then prove its MA-hardness.

4.1 Definition of the SetCSP problem

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 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 $1 \le i \le m$.

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

Definition 4.1 (Set constraint)**.** 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 of sets of strings, $Y(C) = \{Y_1, ..., Y_\ell\}$, where $Y_i \subseteq \{0,1\}^k$ is a set of k-bit strings and $Y_i \cap Y_j = \emptyset$ for all distinct $1 \le i, j \le \ell$.

Definition 4.2 (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 in $Y(C)$, 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, for any string $s \in S$, one can replace its k-bit restriction to the bits $J(C)$, which is a string in some $Y_j \in Y(C)$, with a different k-bit string in Y_j, and the resulting string s' must also be in S.

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 any set S of n-bit strings is *far* from satisfying the collection of set-constraints. 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 $\mathsf{set\text{-}unsat}(\mathcal{C}, S)$. Intuitively, this quantity captures how much we need to modify S in order to satisfy the collection of 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 provide the exact definition of $\mathsf{set\text{-}unsat}(\mathcal{C}, S)$ in Section 4.2.

Definition 4.3 (k-local Set Constraint Satisfaction problem (k-$\mathsf{SetCSP}_\varepsilon$))**.** 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 a non-empty $S \subseteq \{0,1\}^n$ that satisfies all set constraints in $\mathcal{C}$: $\mathsf{set\text{-}unsat}(\mathcal{C}, S) = 0$
No. For all $S \subseteq \{0,1\}^n$, $\mathsf{set\text{-}unsat}(\mathcal{C}, S) \ge \varepsilon(n)$.

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

We present some concepts required for the formal definition of $\textsf{set-unsat}(\mathcal{C}, S)$.

Definition 4.4 (C-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 4.5 (C-Longing strings)**.** Given some set $S \subseteq \{0,1\}^n$ and a set-constraint C, $x \in S$ is a C-longing string with respect to S 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 4.6 (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 4.7 (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 $\mathcal{C}$ and just say that the string is "good".

Given Definitions 4.5 and 4.6 above, it is easy to see that a set of strings $S \subseteq \{0,1\}^n$ satisfies a set-constraint C (by Definition 4.2) 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 4.8 (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 $\textsf{set-unsat}$-value (which we sometimes refer to as the *frustration*) of a set-constraint C with respect to S, is defined by

$$\textsf{set-unsat}(C, S) = \frac{|B_C|}{|S|} + \frac{|L_C|}{|S|} \tag{2}$$

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

$$\textsf{set-unsat}(\mathcal{C}, S) = \frac{1}{m}\sum_{i=1}^{m} \textsf{set-unsat}(C_i, S). \tag{3}$$

We also define the frustration of the set collection $\mathcal{C}$:

$$\textsf{set-unsat}(\mathcal{C}) = \min_{\substack{S \subseteq \{0,1\}^n \\ S \neq \emptyset}} \{\textsf{set-unsat}(\mathcal{C}, S)\}. \tag{4}$$

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

Notice that the normalization factors in Equation (2) guarantees that the $\mathsf{set\text{-}unsat}$-value lies between 0 and 1.[12]

4.3 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 of k-SetCSP in the following way: For each constraint $C_\ell, \ell \in \{1, ..., m\}$ out 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. First, if the original CSP instance is satisfiable, we claim that for any satisfying string s we can define the set $S = \{s\}$ consisting of that single string, and S indeed satisfies the collection of set-constraints defined above. To see this, note that in our case, there is no notion of C-neighbors (See Definition 4.4), since all Y_i's contain only a single string. Hence, there are no longing strings; By the definition of set-unsat (Equation (2)) in this case $\mathsf{set\text{-}unsat}(\mathcal{C}, S) = 0$ 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, which is indeed the case if we pick a satisfying assignment. For the other direction, assume $\mathsf{set\text{-}unsat}(\mathcal{C}, S) = 0$ for some set S. This in particular means that all strings in S are C-good for all set-constraints in $\mathcal{C}$. By definition of our $\mathcal{C}$, this means any string $s \in S$ is a satisfying assignment.

We give now some intuition about the set-unsat quantity (Equation (2)). 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 (2), 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 (3) 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 (3) 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 (2) 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[13]. 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.

[12] 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 $\frac{|B_C|}{|S|} + \frac{|L_C|}{|S|} \le \frac{|S|}{|S|} = 1$.

[13] 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 in this note since it seems most natural.

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

In this subsection, we show the MA-hardness of $\mathsf{SetCSP}_{1/poly}$.

Lemma 4.9. *There exists some polynomial $p(x) = \Theta(1/x^3)$ such that for every inverse polynomial $p' < p$, the problem $\mathsf{SetCSP}_{p'(m)}$ is MA-hard.*

To prove this lemma, we show how to reduce any language $\mathsf{L} \in \mathrm{MA}$ to a 6-SetCSP instance. 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.

4.4.1 Intuition for how set-constraints can check histories

In the celebrated proof of the (classical) Cook-Levin theorem, an instance of an NP-language is mapped to an instance for 3-SAT problem. More precisely, the verifier V of the NP-problem, which runs on an n-bit input string x and a $poly(n)$ bit witness y, is mapped to a 3-SAT formula. To do this, a different variable is assigned to the value of each location of the tape of the Turing machine of V at any time step; these variables are used to keep track of the state of the computation of V (namely what is written on the tape) at the different time steps (see Figure 1 for an example). The formula acts on strings of bits, which can be viewed as assignments to *all* these variables; such an assignment encodes the *entire history* of a single possible computation. The clauses of the Boolean formula check if the history given by the assignment, indeed corresponds to a correct propagation of an *accepted* computation. More precisely, the constraints check that *a*) the assignment to the n Boolean variables at the first time step, associated with the input to the NP-problem, is indeed correct (namely equal to the true input x); *b*) the assignment of the variables corresponding to any two subsequent time steps is consistent with a correct evolution of the appropriate gate in the computation; and *c*) the output bit, namely the value of the output variable in the *last* time step, is indeed *accept*. There exists a valid history which ends with accept (i.e., x is in the language accepted by the verifier), iff the resulting k-CSP is satisfiable; in which case a satisfying assignment encodes a *history* of a correct computation of 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 makes the verification algorithm accept even for no-instances, because the soundness parameter is not 0. 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) most 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.*[14]

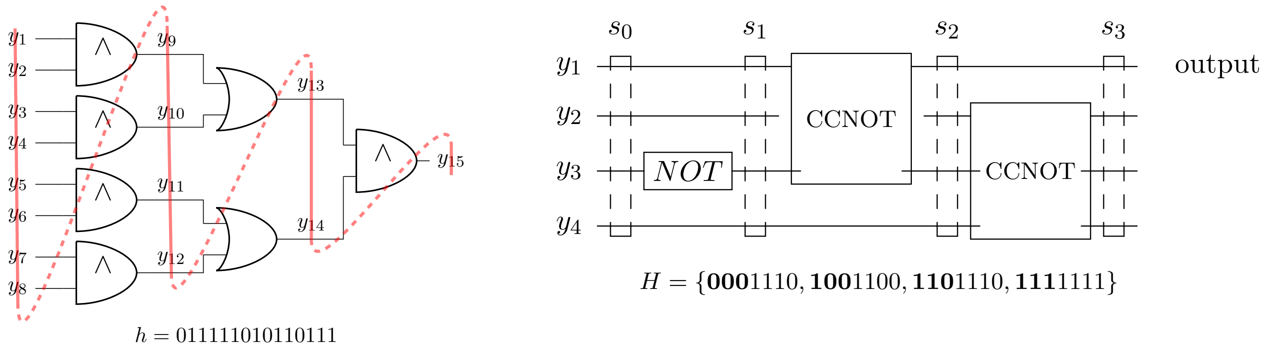
In this case (note that we have still not included random bits in the discussion) we need to show how to create *set-constraints* that verify that the set S really does contain *all* snapshots, and it also needs to verify that S contains nothing else. More precisely, we need to verify that a) the strings 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 (5)).

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

[14] 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.



(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 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 here ensures that a satisfying set of strings contains the snapshots for *all* timesteps of the computation.

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

We next explain the reduction from MA-verification algorithms to 6-SetCSP in Section 4.4.2, and then prove its correctness in Section 4.4.3.

4.4.2 The reduction

We assume, without loss of generality, that the MA verification algorithm is reversible, as described in Section 2.2: 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

$G_i \in \{\mathrm{NOT}, \mathrm{CNOT}, \mathrm{CCNOT}\}$.[15] 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 problem 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 2.2) 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 4.4.1 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:

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

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

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* $j \in [a]$ 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.

[15] 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 $a, b, c \oplus ab$.

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

$$Y(C_j^{rand}) = \{\{00, 01\}, \{10\}, \{11\}\} \text{ and } J(C_j^{rand}) = (0, T+p+a+j). \tag{5}$$

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 C_{t+1}^{prop}- 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}.

4.4.3 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 4.10 (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 4.11 (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.[16]

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

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 A.2. From these 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 $G_{\mathcal{C}}$[17] 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 C.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. □

From the two previous lemmas, we have the following.

Lemma 4.9 (restated). There exists some polynomial $p(x) = \Theta(1/x^3)$ such that for every inverse polynomial $p' < p$, the problem $\mathsf{SetCSP}_{p'(m)}$ is MA-hard.

Proof. It follows directly from Lemma 4.10, together with the fact that, regarding the parameters in Lemma 4.11, we have that $T, q \leq m$. □

[17] Here, we use the same notation as Section 3.1.

5 Reduction from SetCSP to ACAC

In this section we reduce the SetCSP problem to the ACAC, showing the containment of SetCSP in MA and the MA-hardness of ACAC.

Before showing the reduction, we prove a technical lemma that shows how, for a fixed S, the value of $\mathsf{set\text{-}unsat}(\mathcal{C}, S)$ and the number of bad and longing strings for $\mathcal{C}$ are related.

Lemma 5.1. *For some fixed non-empty $S \subset \{0,1\}^n$, let $B_{\mathcal{C}} = \bigcup_i B_{C_i}$ and $L_{\mathcal{C}} = \bigcup_i L_{C_i}$, the union of bad and longing strings for all set-constraints in $\mathcal{C}$, respectively. We have that $\mathsf{set\text{-}unsat}(\mathcal{C}, S) \leq \frac{1}{|S|}(|B_{\mathcal{C}}| + |L_{\mathcal{C}}|)$.*

Proof. By definition of $\mathsf{set\text{-}unsat}(\mathcal{C}, S)$ and $\mathsf{set\text{-}unsat}(C_i, S)$, we have that

$$\begin{aligned}\mathsf{set\text{-}unsat}(\mathcal{C}, S) &= \frac{1}{m}\sum_{i=1}^{m}\mathsf{set\text{-}unsat}(C_i, S) = \frac{1}{m|S|}\sum_{i=1}^{m}|B_{C_i}| + |L_{C_i}| \leq \frac{1}{m|S|}\sum_{i=1}^{m}|B_{\mathcal{C}}| + |L_{\mathcal{C}}| \\ &= \frac{1}{|S|}(|B_{\mathcal{C}}| + |L_{\mathcal{C}}|),\end{aligned}$$

where in the inequality we use the fact that $B_{C_i} \subseteq B_{\mathcal{C}}$ and $L_{C_i} \subseteq L_{\mathcal{C}}$. □

For some inverse-polynomial ε, we consider an instance $\mathcal{C}$ of k-$\mathsf{SetCSP}_{\varepsilon}$. From $\mathcal{C}$, we construct the 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. We can define $C_{G_{\mathcal{C}}}$ that on input $x \in \{0,1\}^n$, outputs all neighbors of x by inspecting all set-constraints of $\mathcal{C}$. Finally, we define C_M as the circuit that on input $x \in \{0,1\}^n$, outputs if x is a bad string for $\mathcal{C}$, again by inspecting all of its set-constraints.

Lemma 5.2 (Reduction from SetCSP to ACAC)**.** *For every ε we have that:*

- *If $\mathcal{C} = (C_1, ..., C_m)$ is a yes-instance of k-$\mathsf{SetCSP}_{\varepsilon}$, then $(C_{G_{\mathcal{C}}}, C_M)$ is a yes-instance of $\mathsf{ACAC}_{\varepsilon/2}$.*
- *If $\mathcal{C} = (C_1, ..., C_m)$ is a no-instance of k-$\mathsf{SetCSP}_{\varepsilon}$, then $(C_{G_{\mathcal{C}}}, C_M)$ is a no-instance of $\mathsf{ACAC}_{\varepsilon/2}$.*

Proof. To prove the first part, we show that a non-empty $S \subseteq \{0,1\}^n$ such that $\mathsf{set\text{-}unsat}(S, \mathcal{C}) = 0$ implies that the connected component of any string $x \in S$ in $G_{\mathcal{C}}$ contains only good strings. To show this, we notice that S is a union of connected components of $G_{\mathcal{C}}$. In this case, any of these connected components imply that $(C_{G_{\mathcal{C}}}, C_M)$ is a yes-instance of ACAC.

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 $\mathsf{set\text{-}unsat}(S, \mathcal{C}) = 0$. We finish this part of the proof by stressing that, by assumption, no elements in S are marked, otherwise there would be a bad string in it.

For the second part, we show that if there is a set of vertices S such that $\partial(S, \overline{S}) < \varepsilon|S|/2$ on $G_{\mathcal{C}}$ and the number of marked elements in S is strictly less than $\varepsilon|S|/2$, then $\mathsf{set\text{-}unsat}(S, \mathcal{C}) < \varepsilon$. In this case, if $\mathcal{C}$ is a no-instance of k-$\mathsf{SetCSP}_{\varepsilon}$, then $(C_{G_{\mathcal{C}}}, C_M)$ must be a no-instance of $\mathsf{ACAC}_{\varepsilon/2}$.

Notice that if there are at most $\varepsilon|S|/2$ edges between S and $\overline{S}$, then there are at most $\varepsilon|S|/2$ vertices in S that are connected to $\overline{S}$ and, by definition of the edges in $G_{\mathcal{C}}$, we have that S has at most $\varepsilon|S|/2$ $\mathcal{C}$-longing strings. We also have that the number of bad strings in S is, by definition, the number of marked elements which is also strictly less than $\varepsilon|S|/2$. Therefore, by Lemma 5.1, we have that $\mathsf{set\text{-}unsat}(\mathcal{C}, S) < \varepsilon$. □

We can now finally prove Theorem 1.1:

Theorem 1.1 (restated). There exists some polynomial $p(x) = \Theta(1/x^3)$ such that for every inverse polynomial $p' < p$, the problems $\mathsf{SetCSP}_{p'(m)}$ and $\mathsf{ACAC}_{p'(m)/2}$ are MA-complete, where m is the size of the SetCSP or ACAC instance.

Proof. From Lemma 4.9 we have that $\mathsf{SetCSP}_{p'}$ is MA-hard[18] and from Corollary 3.5 we have that $\mathsf{ACAC}_{p'/2}$ is in MA.

In Lemma 5.2, we show a reduction $\mathsf{SetCSP}_{p'}$ to $\mathsf{ACAC}_{p'/2}$, which implies, together with the aforementioned results, that $\mathsf{SetCSP}_{p'}$ is in MA and that $\mathsf{ACAC}_{p'/2}$ is MA-hard, finishing the proof. □

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach.* Cambridge University Press, 2009.

[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.

[BBT06] Sergey Bravyi, Arvid J. Bessen, and Barbara M. Terhal. Merlin-arthur games and stoquastic complexity. *arXiv preprint arXiv:0611021*, 2006.

[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.

[CBB10] Elizabeth Crosson, Dave Bacon, and Kenneth R. Brown. Making classical ground-state spin computing fault-tolerant. *Phys. Rev. E*, 82:031106, Sep 2010.

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

[18] Notice that we slightly abuse the notation here: in Lemma 4.9, we define the hardness in respect to the parameter m, the number of clauses; here, we call m the size of the SetCSP instance, which is lower-bounded by its number of clauses.

[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.

[Lad89] Richard E. Ladner. Polynomial space counting problems. *SIAM J. Comput.*, 18(6):1087–1097, 1989.

[LMT00] Klaus-Jörn Lange, Pierre McKenzie, and Alain Tapp. Reversible space equals deterministic space. *J. Comput. Syst. Sci.*, 60(2):354–367, 2000.

[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 Complexity classes

Definition A.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.

Definition A.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 $x \in A_{\text{yes}}$, there exists a witness y such that R accepts (x, y) with probability 1.
Soundness. If $x \in A_{\text{no}}$, for any witness y, R accepts (x, y) with probability at most $\frac{1}{3}$.

B 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 Boolean circuits, which are usually described using AND, OR and NOT gates [AB09]. 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 the output string of the gate is a one-to-one function of the input string, and thus an inverse gate exists. Clearly, we can also invert the entire circuit made of such gates by running all inverses of the gates in reverse order.

Boolean circuits consisting of the *universal* gateset $\{\text{NOT}, \text{CNOT}, \text{CCNOT}\}$,[19] can be made reversible, by making use of additional auxiliary bits initialized to 0. It is folklore that the overhead for this conversion is linear in the number of gates. We will assume all verifiers considered are reversible Boolean circuits.

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.

[19] 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.

C 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 C.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 $p^T_{u,v} \in [\pi_G(v) - \varepsilon, \pi_G(v) + \varepsilon]$.* [20]

We now prove Lemma 3.4

Lemma 3.4 (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

[20] 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 C.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.

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

$$\phi_{G'}(A') = \frac{|\partial_{G'}(A')|}{vol_{G'}(A')} = \frac{|\partial_G(A')|}{\sum_{v\in A'} 2d_G(v)} \geq \frac{\delta|A'|}{d|A'|} = \frac{\delta}{d}. \tag{6}$$

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

$$\phi_{G'}(S) = \frac{w(\partial_{G'}(S))}{vol_{G'}(S)} \geq \frac{w(\partial_{G'}(\overline{S}))}{vol_{G'}(\overline{S})} = \frac{|\partial_G(\overline{S})|}{\sum_{v\in\overline{S}} d_G(v)} \geq \frac{|\partial_G(\overline{S})|}{d|\overline{S}|} \geq \frac{\delta|\overline{S}|}{d|\overline{S}|} = \frac{\delta}{d}, \tag{7}$$

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 (6) and (7), 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 C.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. □

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

D PSPACE-completeness of CCC

In this section we give a sketch of proof that the CCC problem is PSPACE-complete.

Lemma D.1. CCC *is in* PSPACE*.*

Proof sketch. In order to prove containment, we can follow two steps

1. A variant of CCC, called FCCC, is in PSPACE.
2. The containment of FCCC in PSPACE implies the containment of CCC in PSPACE.

The problem FCCC consists of deciding if the connected component of a fixed node contains only unmarked elements. More concretely, it corresponds to the language

$$\{(x, C_G, C_M) : \text{the connected component of } x \text{ in the graph } G \text{ contains only unmarked elements}\},$$

where C_G and C_M are defined as in CCC. We argue that FCCC is in PPSPACE,[21] and since PPSPACE=PSPACE [Lad89], this proves the first claim. The PPSPACE algorithm consists of a 2^n-step random walk on G starting from x which rejects iff a marked element is encountered. If the connected component of x has no marked elements, so the algorithm never rejects, whereas for no-instances, the random walk finds the marked element with non-zero probability.

We can devise a polynomial-space algorithm for CCC that iterates over all $x \in \{0,1\}^n$ and accept iff for at least one of such x's, (x, C_G, C_M) is a yes-instance of FCCC (which can be decided in PSPACE as described before). □

Lemma D.2. CCC *is* PSPACE*-hard.*

Proof sketch. Let L be a language in PSPACE. Since we can assume that any problem in PSPACE can also be solved in polynomial space by a *reversible* Turing machine [LMT00], we fix the reversible Turing Machine M that decides if x belongs to L in $p(|x|)$-space, for some polynomial p. A *configuration* of M consists of its internal state, the content of its tape and the position of the tape-head. For some configuration w, we denote $pred(w)$ as the preceding configuration of w and $succ(w)$ as its following configuration (and both can be efficiently computed from w since M is reversible).

In the reduction to CCC, we set $n = p(|x|) + S$, where $S = O(p(|x|))$ is the amount of memory to store a snapshot of the computation of M (meaning the current state of the Turing machine, the tape and the position of the head). Each string $y \in \{0,1\}^n$ can be split in two parts: the first one is the "clock"-substring and the second one is the "working"-substring. The clock substring represents, in binary, a value in $t \in [2^{p(|x|)}]$ (notice that M runs in time at most $2^{p(n)}$) and the working substring represents an arbitrary state of the Turing Machine at time t.

We define then instance (C_G, C_M) of CCC as follows:

- On $t \,||\, w \in \{0,1\}^n$, C_G outputs the string $t-1 \,||\, pred(w)$ and $t+1 \,||\, succ(w)$ (if $t = 0$ or $t = 2^n$, C_G only outputs the meaningful neighbors).

[21] PPSPACE is the complexity class of randomized algorithms that run in polynomial space and non-zero completeness/soundness gap

- On $t \,||\, w \in \{0,1\}^n$, C_M outputs 1 if $t = 0$ and w does not consist of the state of M at the beginning of the computation or if $t = 2^{p(|x|)}$ and w does not consist of the state of M that would lead to acceptance.

It is not hard to see that if $x \in L$, then the connected component of $0 \,||\, x$ is clean, whereas if $x \notin L$, all connected components have at least one marked element. □