Improved NP-inapproximability for 2-variable linear equations

(full version)

Johan Håstad *
johanh@csc.kth.se

Sangxia Huang*
sangxia@csc.kth.se

Rajsekar Manokaran*
rajsekar@csc.kth.se

Ryan O'Donnell †
odonnell@cs.cmu.edu

John Wright†
jswright@cs.cmu.edu

July 3, 2014

Abstract

An instance of the 2-Lin(2) problem is a system of equations of the form "$x_i + x_j = b \pmod 2$". Given such a system in which it's possible to satisfy all but an ϵ fraction of the equations, we show it is NP-hard to satisfy all but a $C\epsilon$ fraction of equations, for any $C < \frac{11}{8} = 1.375$ (and any $0 < \epsilon \leq \frac{1}{8}$). The previous best result, standing for over 15 years, had $\frac{5}{4}$ in place of $\frac{11}{8}$. Our result provides the best known NP-hardness even for the Unique-Games problem, and it also holds for the special case of Max-Cut. The precise factor $\frac{11}{8}$ is unlikely to be best possible; we also give a conjecture concerning analysis of Boolean functions which, if true, would yield a larger hardness factor of $\frac{3}{2}$.

Our proof is by a modified gadget reduction from a pairwise-independent predicate. We also show an inherent limitation to this type of gadget reduction. In particular, any such reduction can never establish a hardness factor C greater than 2.54. Previously, no such limitations on gadget reductions was known.

*School of Computer Science and Communication, KTH Royal Institute of Technology.

†Department of Computer Science, Carnegie Mellon University. Supported by NSF grants CCF-0747250 and CCF-1116594 and a grant from the MSR–CMU Center for Computational Thinking.

1 Introduction

The well known constraint satisfaction problem (CSP) 2-Lin(q) is defined as follows: Given n variables $x_1, \ldots, x_n$, as well as a system of equations (constraints) of the form "$x_i - x_j = b \pmod q$" for constants $b \in \mathbb{Z}_q$, the task is to assign values from $\mathbb{Z}_q$ to the variables so that there are as few unsatisfied constraints as possible. It is known [KKMO07, MOO10] that, from an approximability standpoint, this problem is equivalent to the notorious Unique-Games problem [Kho02]. The special case of $q = 2$ is particularly interesting and can be equivalently stated as follows: Given a "supply graph" G and a "demand graph" H over the same set of vertices V, partition V into two parts so as to minimize the total number of cut supply edges and uncut demand edges. The further special case when the supply graph G is empty (i.e., every equation is of the form $x_i - x_j = 1 \pmod 2$) is equivalent to the Max-Cut problem.

Let's say that an algorithm guarantees an (ϵ, ϵ')-*approximation* if, given any instance in which the best solution falsifies at most an ϵ-fraction of the constraints, the algorithm finds a solution falsifying at most an ϵ'-fraction of the constraints. If an algorithm guarantees $(\epsilon, C\epsilon)$-approximation for every ϵ then we also say that it is a *factor-C approximation*. To illustrate the notation we recall two simple facts. On one hand, for each fixed q, there is a trivial greedy algorithm which $(0, 0)$-approximates 2-Lin(q). On the other hand, (ϵ, ϵ)-approximation is NP-hard for every $0 < \epsilon < \frac{1}{q}$; in particular, factor-1 approximation is NP-hard.

We remark here that we are prioritizing the so-called "Min-Deletion" version of the 2-Lin(2) problem. We feel it is the more natural parameterization. For example, in the more traditional "Max-2-Lin(2)" formulation, the discrepancy between known algorithms and NP-hardness involves two quirky factors, 0.878 and 0.912. However, this disguises what we feel is the really interesting question — the same key open question that arises for the highly analogous Sparsest-Cut problem: Is there an efficient $(\epsilon, O(\epsilon))$-approximation, or even one that improves on the known $(\epsilon, O(\sqrt{\log n})\epsilon)$- and $(\epsilon, O(\sqrt{\epsilon}))$-approximations?

The relative importance of the "Min-Deletion" version is even more pronounced for the 2-Lin(q) problem. As we describe below, this version of the problem is essentially equivalent to the highly notorious Unique-Games problem. By way of contrast, the traditional maximization approximation factor measure for Unique-Games is not particularly interesting — it's known [FR04] that there is no constant-factor approximation for "Max-Unique-Games", but this appears to have no relevance for the Unique Games Conjecture.

1.1 History of the problem

No efficient $(\epsilon, O(\epsilon))$-approximation algorithm for 2-Lin(2) is known. The best known efficient approximation guarantee with no dependence on n dates back to the seminal work of Goemans and Williamson:

Theorem 1.1. *([GW94].) There is a polynomial-time $(\epsilon, \frac{2}{\pi}\sqrt{\epsilon} + o(\epsilon))$-approximation algorithm for* 2-Lin(2).

Allowing the approximation to depend on n, we have the following result building on [ARV04]:

Theorem 1.2. *([ACMM05].) There is a polynomial-time factor-$O(\sqrt{\log n})$ approximation for* 2-Lin(2).

Generalizing Theorem 1.1 to 2-Lin(q), we have the following result of Charikar, Makarychev, and Makarychev:

Theorem 1.3. *([CMM06].) There is a polynomial time $(\epsilon, C_q\sqrt{\epsilon})$-approximation for* 2-Lin(q) *(and indeed for* Unique-Games*), for a certain $C_q = \Theta(\sqrt{\log q})$.*

The question of whether or not this theorem can be improved is known to be essentially equivalent to the influential Unique Games Conjecture of Khot [Kho02]:

Theorem 1.4. *The Unique Games Conjecture implies ([KKMO07, MOO10]) that improving on Theorems 1.1, 1.3 is* NP*-hard. On the other hand ([Rao11]), if there exists $q = q(\epsilon)$ such that $(\epsilon, \omega(\sqrt{\epsilon}))$-approximating* 2-Lin($q$) *is* NP*-hard then the Unique Games Conjecture holds.*

The recent work of Arora, Barak, and Steurer has also emphasized the importance of subexponential-time algorithms in this context:

Theorem 1.5. *([ABS10].) For any $\beta \geq \frac{\log\log n}{\log n}$ there is a $2^{O(qn^\beta)}$-time algorithm for $(\epsilon, O(\beta^{-3/2})\sqrt{\epsilon})$-approximating* 2-Lin($q$)*. For example, there is a constant $K < \infty$ and an $O(2^{n^{0.001}})$-time algorithm for $(\epsilon, K\sqrt{\epsilon})$-approximating* 2-Lin($q$) *for any $q = n^{o(1)}$.*

Finally, we remark that there is an *exact* algorithm for 2-Lin(2) running in time roughly 1.73^n. [Wil05].

The known NP-hardness results for 2-Lin(q) are rather far from the known algorithms. It follows easily from the PCP Theorem that for any q, there exists $C > 1$ such that factor-C approximation of 2-Lin(q) is NP-hard. However, getting an explicit value for C has been a difficult task. In 1995, Bellare, Golreich, and Sudan [BGS95] introduced the Long Code testing technique, which let them prove NP-hardness of approximating 2-Lin(2) to factor of roughly 1.02. Around 1997, Håstad [Hås97] gave an optimal inapproximability result for the 3-Lin(2) problem; combining this with the "automated gadget" results of Trevisan et al. [TSSW00] allowed him to establish NP-hardness of factor-C approximation for any $C < \frac{5}{4}$. By including the "outer PCP" results of Moshkovitz and Raz [MR10] we may state the following more precise theorem:

Theorem 1.6. *([Hås97].) Fix any $C < \frac{5}{4}$. Then it is* NP*-hard to $(\epsilon, C\epsilon)$-approximate* 2-Lin(2) *(for any $0 < \epsilon \leq \epsilon_0 = \frac{1}{4}$). In fact ([MR10]), there is a reduction with quasilinear blowup; hence $(\epsilon, C\epsilon)$-approximation on size-N instances requires $2^{N^{1-o(1)}}$ time assuming the Exponential Time Hypothesis (ETH).*

Since 1997 there had been no improvement on this hardness factor of $\frac{5}{4}$, even for the (presumably much harder) 2-Lin(q) problem. We remark that Håstad [Hås97] showed the same hardness result even for Max-Cut (albeit with a slightly smaller ϵ_0) and that O'Donnell and Wright [OW12] showed the same result for 2-Lin(q) (even with a slightly larger ϵ_0, namely $\epsilon_0 \to \frac{1}{2}$ as $q \to \infty$).

1.2 Our results and techniques

In this work we give the first known improvement to the factor-$\frac{5}{4}$ NP-hardness for 2-Lin(2) from [Hås97]:

Theorem 1.7. *Fix any $C < \frac{11}{8}$. Then it is* NP*-hard to $(\epsilon, C\epsilon)$-approximate* 2-Lin(2) *(for any $0 < \epsilon \leq \epsilon_0 = \frac{1}{8}$). Furthermore, the reduction takes* 3-Sat *instances of size n to* 2-Lin(2) *instances of size $O(n^7)$; hence $(\epsilon, C\epsilon)$-approximating* 2-Lin(2) *instances of size N requires $2^{\Omega(N^{1/7})}$ time assuming the ETH.*

This theorem is proven in Section 3, wherein we also note that the same theorem holds in the special case of Max-Cut (albeit with some smaller, inexplicit value of ϵ_0).

Our result is a gadget reduction from the "7-ary Hadamard predicate" CSP, for which Chan [Cha13] recently established an optimal NP-inapproximability result. In a sense our Theorem 1.7 is a direct generalization of Håstad's Theorem 1.6, which involved an optimal gadget reduction from the "3-ary Hadamard predicate" CSP, namely 3-Lin(2). That said, we should emphasize some obstacles that prevented this result from being obtained 15 years ago.

First, we employ Chan's recent approximation-resistance result for the 7-ary Hadamard predicate. In fact, what's crucial is not its approximation-resistance, but rather the stronger fact that it's a *useless* predicate, as defined in the recent work [AH12]. That is, given a nearly-satisfiable instance of the CSP, it's NP-hard to assign values to the variables so that the distribution on constraint 7-tuples is noticeably different from the uniform distribution.

Second, although in principle our reduction fits into the "automated gadget" framework of Trevisan et al. [TSSW00], in practice it's completely impossible to find the necessary gadget automatically, since it would involve solving a linear program with 2^{256} variables. Instead we had to construct and analyze our gadget by hand. On the other hand, by also constructing an appropriate LP dual solution, we are able to show the following in Section 4:

Theorem 1.8. *(Informally stated.) Our gadget achieving factor-$\frac{11}{8}$ NP-hardness for* 2-Lin(2) *is* optimal *among gadget reductions from Chan's 7-ary Hadamard predicate hardness.*

In spite of Theorem 1.8, it seems extremely unlikely that factor-$\frac{11}{8}$ NP-hardness for 2-Lin(2) is the end of the line. Indeed, we view Theorem 1.7 as more of a "proof of concept" illustrating that the longstanding factor-$\frac{5}{4}$ barrier can be broken; we hope to see further improvements in the future. In particular, in Section 5 we present a candidate NP-hardness reduction from high-arity useless CSPs that we believe may yield NP-hardness of approximating 2-Lin(2) to any factor no larger than $\frac{3}{2}$. The analysis of this reduction eventually depends on a certain conjecture regarding analysis of Boolean functions that we were unable to resolve; thus we leave it as an open problem.

Finally, in Section 6 we show an inherent limitation of the method of gadget reductions from pairwise-independent predicates. We prove that such reductions can never establish an NP-hardness factor better than $\frac{1}{1-e^{-1/2}} \approx 2.54$ for $(\varepsilon, C\varepsilon)$-approximation of 2-Lin(2). We believe that this highlights a serious bottleneck in obtaining an inapproximability result matching the performance of algorithms for this problem as most optimal NP-inapproximability results involve pairwise-independent predicates.

2 Preliminaries

Definition 2.1. Given $x, y \in \{-1,1\}^n$, the *Hamming distance* between x and y, denoted $d_H(x,y)$, is the number of coordinates i where x_i and y_i differ. Similarly, if $f, g : V \to \{-1,1\}$ are two functions over a variable set V, then the Hamming distance $d_H(f,g)$ between them is the number of inputs x where $f(x)$ and $g(x)$ disagree.

Definition 2.2. A *predicate* on n variables is a function $\phi : \{-1,1\}^n \to \{0,1\}$. We say that $x \in \{-1,1\}^n$ *satisfies* ϕ if $\phi(x) = 1$ and otherwise that it *violates* ϕ.

Definition 2.3. Given a predicate $\phi : \{-1,1\}^n \to \{0,1\}$, $\mathsf{Sat}(\phi)$ is the set of satisfying assignments.

Definition 2.4. A set $S \subseteq \{-1,1\}^n$ is a *balanced pairwise-independent subgroup* if it satisfies the following properties:

1. S forms a group under bitwise multiplication.

2. If $\boldsymbol{x}$ is selected from S uniformly at random, then $\mathbf{Pr}[\boldsymbol{x}_i = 1] = \mathbf{Pr}[\boldsymbol{x}_i = -1] = \frac{1}{2}$ for any $i \in [n]$. Furthermore, $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are independent for any $i \neq j$.

A predicate $\phi : \{-1,1\}^n \to \{0,1\}$ *contains a balanced pairwise-independent subgroup* if there exists a set $S \subseteq \mathsf{Sat}(\phi)$ which is a balanced pairwise-independent subgroup.

Definition 2.5. For a subset $S \subseteq [n]$, the parity function $\chi_S : \{-1,1\}^n \to \{-1,1\}$ is defined as $\chi_S(x) := \prod_{i \in S} x_i$.

Definition 2.6. The Had_k predicate has $2^k - 1$ input variables, one for each nonempty subset $S \subseteq [k]$. The input string $\{x_S\}_{\emptyset \neq S \subseteq [k]}$ satisfies Had_k if for each S, $x_S = \chi_S(x)$.

Fact 2.7. *The Had_k predicate contains a balanced pairwise-independent subgroup. (In fact, the whole set $\mathsf{Sat}(\mathsf{Had}_k)$ is a balanced pairwise-independent subgroup.)*

Given a predicate $\phi : \{-1,1\}^n \to \{0,1\}$, an instance $\mathcal{I}$ of the Max-ϕ CSP is a variable set V and a distribution of ϕ-constraints on these variables. To sample a constraint from this distribution, we write $\mathcal{C} \sim \mathcal{I}$, where $\mathcal{C} = ((x_1, b_1), (x_2, b_2), \ldots, (x_n, b_n))$. Here the x_i's are in V and the b_i's are in $\{-1,1\}$. An assignment $A : V \to \{-1,1\}$ satisfies the constraint $\mathcal{C}$ if

$$\phi\left(b_1 \cdot A(x_1), b_2 \cdot A(x_2), \ldots, b_n \cdot A(x_n)\right) = 1.$$

We define several measures of assignments and instances.

Definition 2.8. The *value* of A on $\mathcal{I}$ is just $\mathsf{val}(A;\mathcal{I}) := \mathbf{Pr}_{\mathcal{C} \sim \mathcal{I}}[A \text{ satisfies } \mathcal{C}]$, and the value of the instance $\mathcal{I}$ is $\mathsf{val}(\mathcal{I}) := \max_{\text{assignments } A} \mathsf{val}(A;\mathcal{I})$. We define $\mathsf{uval}(A;\mathcal{I}) := 1 - \mathsf{val}(A;\mathcal{I})$ and similarily $\mathsf{uval}(\mathcal{I})$.

Definition 2.9. Let $(=) : \{-1,1\}^2 \to \{0,1\}$ be the equality predicate, i.e. $(=)(b_1, b_2) = 1$ iff $b_1 = b_2$ for all $b_1, b_2 \in \{-1,1\}$. We will refer to the Max-$(=)$ CSP as the *Max*-2-Lin(2) CSP. Any constraint $\mathcal{C} = ((x_1, b_1), (x_2, b_2))$ in a Max-2-Lin(2) instance tests "$x_1 = x_2$" if $b_1 \cdot b_2 = 1$, and otherwise tests "$x_1 \neq x_2$".

Typically, a hardness of approximation result will show that given an instance $\mathcal{I}$ of the Max-ϕ problem, it is NP-hard to tell whether $\mathsf{val}(\mathcal{I}) \geq c$ or $\mathsf{val}(\mathcal{I}) \leq s$, for some numbers $c > s$. A stronger notion of hardness is *uselessness*, first defined in [AH12], in which in the second case, not only is $\mathsf{val}(\mathcal{I})$ small, but any assignment to the variables A appears "uniformly random" to the constraints. To make this formal, we will require a couple of definitions.

Definition 2.10. Given two probability distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ on some set S, the total variation distance d_{TV} between them is defined to be $d_{TV}(\mathcal{D}_1, \mathcal{D}_2) := \sum_{e \in S} \frac{1}{2} |\mathcal{D}_1(e) - \mathcal{D}_2(e)|$.

Definition 2.11. Given a Max-ϕ instance $\mathcal{I}$ and an assignment A, denote by $\mathcal{D}(A, \mathcal{I})$ the distribution on $\{-1,1\}^n$ generated by first sampling $((x_1, b_1), \ldots, (x_n, b_n)) \sim \mathcal{I}$ and then outputting $(b_1 \cdot A(x_1), \ldots, b_n \cdot A(x_n))$.

The work of [Cha13] showed uselessness for a wide range of predicates, including the Had_k predicate.

Theorem 2.12 ([Cha13])**.** *Let $\phi : \{-1,1\}^n \to \{0,1\}$ contain a balanced pairwise-independent subgroup. For every $\epsilon > 0$, given an instance $\mathcal{I}$ of Max-ϕ, it is NP-hard to distinguish between the following two cases:*

- *(Completeness):* $\mathsf{val}(\mathcal{I}) \geq 1 - \epsilon$.
- *(Soundness): For every assignment A, $d_{TV}(\mathcal{D}(A, \mathcal{I}), \mathcal{U}_n) \leq \epsilon$, where $\mathcal{U}_n$ is the uniform distribution on $\{-1,1\}^n$.*

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & 1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & 1 & -1 & -1 \\ 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ -1 & 1 & 1 & -1 & -1 & 1 & -1 \\ -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 & -1 \end{bmatrix}$$

Figure 1: The Had_3-matrix. The rows are the satisfying assignments of Had_3.

2.1 Gadgets

The work of Trevisan et al [TSSW00] gives a generic methodology for constructing gadget reductions between two predicates. In this section, we review this with an eye towards our eventual Had_k-to-2-Lin(2) gadgets.

Suppose $\phi : \{-1,1\}^n \to \{0,1\}$ is a predicate one would like to reduce to another predicate $\psi : \{-1,1\}^m \to \{0,1\}$. Set $K := |\mathsf{Sat}(\phi)|$. We begin by arranging the elements of $\mathsf{Sat}(\phi)$ as the rows of a $K \times n$ matrix, which we will call the *ϕ-matrix*. An example of this is done for the Had_3 predicate in Figure 1. The columns of this matrix are elements of $\{-1,1\}^K$. Naming this set $V := \{-1,1\}^K$, we will think of V as the set of possible variables to be used in a gadget reduction from ϕ to ψ. One of the contributions of [TSSW00] was to show that the set V is sufficient for any such gadget reduction, and that any gadget reduction with more than 2^K variables has redundant variables which can be eliminated.

Of these variables, the n variables found as the columns of the ϕ-matrix are special; they correspond to n of the variables in the original ϕ instance and are therefore called *generic primary* variables. We will call them $v_1, v_2, \ldots, v_n$, where they are ordered by their position in the ϕ-matrix. The remaining variables are called generic *auxiliary* variables. For example, per Figure 1, $(1,1,1,1,-1,-1,-1,-1)$ and $(1,-1,-1,1,-1,1,1,-1)$ are generic primary variables in any gadget reducing from ϕ, but $(-1,-1,1,-1,1,-1,1,-1)$ is always a generic auxiliary variable.

On top of the variables V will be a distribution of ψ constraints. As a result, a gadget $\mathcal{G}$ is just an instance of the Max-ψ CSP using the variable set V. As above, we will associate $\mathcal{G}$ with the distribution of ψ constraints and write $\mathcal{C} \sim \mathcal{G}$ to sample a constraint from this distribution. Given an assignment $A : V \to \{0,1\}$, the goal is for $\mathcal{G}$ to be able to detect whether the values A assigns to the generic primary variables satisfy the ϕ predicate. For shorthand, we will say that A *satisfies* ϕ when

$$\phi\left(A(v_1), A(v_2), \ldots, A(v_n)\right) = 1.$$

On the other hand, A *fails to satisfy* ϕ when this expression evaluates to 0. Of all assignments, we are perhaps most concerned with the *dictator* assignments. The i-th dictator assignment, written $d_i : \{-1,1\}^K \to \{-1,1\}$, is defined so that $d_i(x) = x_i$ for all $x \in \{-1,1\}^K$. The following fact shows why the dictator assignments are so important:

Fact 2.13. *Each dictator assignment d_i satisfies ϕ.*

Proof. The string $((v_1)_i, (v_2)_i, \ldots, (v_n)_i)$ is the i-th row of the ϕ-matrix, which, by definition, satisfies ϕ. □

At this point, we can now give the standard definition of a gadget. Typically, one constructs a gadget so that the dictator assignments pass with high probability, whereas every assignment which fails to satisfy ϕ passes with low probability. This is formalized in the following definition, which is essentially from [TSSW00]:

Definition 2.14 (Old definition)**.** A (c, s)-*generic gadget* reducing Max-ϕ to Max-ψ is a gadget $\mathcal{G}$ satisfying the following properties:

- (Completeness): For every dictator assignment d_i, $\mathsf{uval}(d_i; \mathcal{G}) \leq c$.
- (Soundness): For any assignment A which fails to satisfy ϕ, $\mathsf{uval}(A; \mathcal{G}) \geq s$.

We use uval as our focus is on the deletion version of 2-Lin(2). We include the word *generic* in this definition to distinguish it from the specific type of gadget we will use to reduce Had_k to 2-Lin(2). See Section 2.3 for details.

This style of gadget reduction is appropriate for the case when one is reducing from a predicate for which one knows an inapproximability result and nothing else. However, in our case we are reducing from predicates containing a balanced pairwise-independent subgroup, and Chan [Cha13] has shown *uselessness* for this class of predicates (see Theorem 2.12). As a result, we can relax the (Soundness) condition in Definition 2.14; when reducing from this class of predicates, it is sufficient to show that this (Soundness) condition holds for *distributions* of assignments which *appear random on the generic primary variables*. In the following paragraph we expand on what this means.

Denote by $\mathcal{A}$ a distribution over assignments A. The value of $\mathcal{A}$ is just the average value of an assignment drawn from $\mathcal{A}$, i.e. $\mathsf{val}(\mathcal{A}; \mathcal{G}) := \mathbf{E}_{A \sim \mathcal{A}} \mathsf{val}(A; \mathcal{G})$, and similarly for $\mathsf{uval}(\mathcal{A}; \mathcal{G})$. We say that $\mathcal{A}$ is *random on the generic primary variables* if the tuple

$$(A(v_1), A(v_2), \ldots, A(v_n))$$

is, over a random $A \sim \mathcal{A}$, distributed as a uniformly random element of $\{-1, 1\}^n$.

Definition 2.15. Denote by $\mathsf{R}^{gen}(\phi)$ the set of distributions which are random on the generic primary variables.

Our key definition is the following, which requires that our gadget only does well against distributions in $\mathsf{R}^{gen}(\phi)$.

Definition 2.16 (New definition)**.** A (c, s)-*generic gadget* reducing Max-ϕ to Max-ψ is a gadget $\mathcal{G}$ satisfying the following properties:

- (Completeness): For every dictator assignment d_i, $\mathsf{uval}(d_i; \mathcal{G}) \leq c$.
- (Soundness): For any $\mathcal{A} \in \mathsf{R}^{gen}(\phi)$, $\mathsf{uval}(\mathcal{A}; \mathcal{G}) \geq s$.

The following proposition is standard, and we sketch its proof for completeness.

Proposition 2.17. *Suppose there exists a (c, s)-generic gadget reducing Max-ϕ to Max-ψ, where Max-ϕ is any predicate containing a balanced pairwise-independent subgroup. Then for all $\epsilon > 0$, given an instance $\mathcal{I}$ of Max-ψ, it is NP-hard to distinguish between the following two cases:*

- *(Completeness):* $\mathsf{uval}(\mathcal{I}) \leq c + \epsilon$.
- *(Soundness):* $\mathsf{uval}(\mathcal{I}) \geq s - \epsilon$.

Proof sketch. Let $\mathcal{I}$ be an instance of the Max-ϕ problem produced via Theorem 2.12. To dispense with some annoying technicalities, we will assume that every constraint $\mathcal{C}$ in the support of $\mathcal{I}$ is of the form $\mathcal{C} = ((x_1, 1), \ldots, (x_n, 1))$. Construct an instance $\mathcal{I}'$ of Max-ψ as follows: for each constraint $\mathcal{C} = ((x_1, 1), \ldots, (x_n, 1))$ in the support of $\mathcal{I}$, add in a copy of $\mathcal{G}$ — call it $\mathcal{G}_{\mathcal{C}}$ — whose total weight is scaled down so that it equals the weight of $\mathcal{C}$. Further, identify the primary variables $v_1, \ldots, v_n$ of $\mathcal{G}_{\mathcal{C}}$ with the variables $x_1, \ldots, x_n$.

Completeness: In this case, there exists an assignment A to the variables of $\mathcal{I}$ which violates at most an ϵ-fraction of the constraints. We will extend this to an assignment for all the variables of $\mathcal{I}'$ as follows: for any constraint $\mathcal{C} = ((x_1, 1), \ldots, (x_n, 1))$ which A satisfies, there is some dictator assignment to the variables of $\mathcal{G}_{\mathcal{C}}$ which agrees with A on the primary variables $v_1, \ldots, v_n$. Set A to also agree with this dictator assignment on the auxiliary variables in $\mathcal{G}_{\mathcal{C}}$. Regardless of how A is extended in the remaining $\mathcal{G}_{\mathcal{C}}$'s, it now labels a $(1 - \epsilon)$-fraction of the $\mathcal{G}$ gadgets in $\mathcal{I}'$ with a dictator assignment, meaning that $\mathsf{uval}(A; \mathcal{I}') \leq (1 - \epsilon) \cdot c + \epsilon \cdot 1 \leq c + \epsilon$.

Soundness: Let A be an assignment to the variables in $\mathcal{I}'$. Consider the distribution $\mathcal{A}$ of assignments to the gadget $\mathcal{G}$ generated as follows: sample $\mathcal{C} \sim \mathcal{I}$ and output the restriction of A to the variables of $\mathcal{G}_{\mathcal{C}}$. Because the distribution $(A(x_1), \ldots, A(x_n))$ is ϵ-far from uniform in total variation distance, $\mathcal{A}$ is ϵ-far in total variation distance from some distribution $\mathcal{A}' \in \mathsf{R}^{gen}(\phi)$. As a result, $\mathsf{uval}(\mathcal{A}; \mathcal{G}) \geq \mathsf{uval}(\mathcal{A}'; \mathcal{G}) - \epsilon \geq s - \epsilon$. But then $\mathsf{uval}(\mathcal{A}; \mathcal{G}) = \mathsf{uval}(A; \mathcal{I})$, which is therefore bounded below by $s - \epsilon$. □

2.2 Reducing into 2-Lin(2)

In this section, we consider gadgets which reduce into the 2-Lin(2) predicate. We show several convenient simplifying assumptions that can be made in this case.

Definition 2.18. An assignment $A : \{-1, 1\}^K \to \{-1, 1\}$ is *folded* if $A(x) = -A(-x)$ for all $x \in \{-1, 1\}^K$. Here $-x$ is the bitwise negation of x. In addition, a distribution $\mathcal{A}$ is folded if every assignment in its support is folded.

The following proposition shows that when designing a gadget which reduces into 2-Lin(2), it suffices to ensure that its (Soundness) condition holds for folded distributions. The proof is standard.

Proposition 2.19. *For some predicate ϕ, suppose $\mathcal{G}$ is a gadget reducing Max-ϕ to Max-2-*Lin*(2) which satisfies the following two conditions:*

- *(Completeness): For every dictator assignment d_i, $\mathsf{uval}(d_i; \mathcal{G}) \leq c$.*
- *(Soundness): For any* folded *$\mathcal{A} \in \mathsf{R}^{gen}(\phi)$, $\mathsf{uval}(\mathcal{A}; \mathcal{G}) \geq s$.*

*Then there exists a (c, s)-generic gadget reducing Max-ϕ to Max-2-*Lin*(2).*

Proof. For each pair of antipodal points x and $-x$ in $\{-1, 1\}^K$, pick one (say, x) arbitrarily, and set

$$\mathsf{canon}(x) := \mathsf{canon}(-x) := x.$$

This is the canonical variable associated to x and $-x$. The one constraint is that if either x or $-x$ is one of the generic primary variables, then it should be chosen as the canonical variable associated to x and $-x$. Now, let $\mathcal{G}'$ be the gadget whose constraints are sampled as follows:

1. Sample a constraint $A(x_1) \cdot A(x_2) = b$ from $\mathcal{G}$.
2. For $i \in \{1, 2\}$, set $b_i = 1$ if $\mathsf{canon}(x_i) = x_i$ and $b_i = -1$ otherwise.
3. Output the constraint $A(\mathsf{canon}(x_1)) \cdot A(\mathsf{canon}(x_2)) = b \cdot b_1 \cdot b_2$.

We claim that $\mathcal{G}'$ is a (c, s)-gadget reducing Max-ϕ to Max-2-Lin(2). To see this, set $\mathsf{is\text{-}canon}(x)$ to be 1 if $\mathsf{canon}(x) = x$ and (-1) otherwise. Then the probability that an assignment A fails on $\mathcal{G}'$ is the same as the probability that the assignment $A'(x) := \mathsf{is\text{-}canon}(x) \cdot A(\mathsf{canon}(x))$ fails on $\mathcal{G}$. For any dictator function d_i, $d_i(x) = \mathsf{is\text{-}canon}(x) \cdot d_i(\mathsf{canon}(x))$ for all x. Therefore, d_i fails $\mathcal{G}'$ with probability c. Next, it is easy to see that for any assignment A, A' is folded and, due to our restriction on $\mathsf{canon}(\cdot)$, A' agrees with A on the generic primary variables. Thus, given a distribution $\mathcal{A} \in \mathsf{R}^{gen}(\phi)$, $\mathcal{A}$ fails on $\mathcal{G}'$ with the same probability that some folded distribution in $\mathsf{R}^{gen}(\phi)$ fails on $\mathcal{G}$, which is at least s. □

Proposition 2.20. *For fixed values of c and s, let $\mathcal{G}$ be a gadget satisfying the (Completeness) and (Soundness) conditions in the statement of Proposition 2.19. Then there exists another gadget satisfying these conditions which only uses equality constraints.*

Proof. Let $\mathcal{G}'$ be the gadget which replaces each constraint in $\mathcal{G}$ of the form $x \neq y$ with the constraint $x = -y$. If A is a folded assignment,

$$A(x) \neq A(y) \iff A(x) = A(-y).$$

Thus, for every folded assignment A, $\mathsf{val}(A; \mathcal{G}) = \mathsf{val}(A, \mathcal{G}')$. As the (Completeness) and (Soundness) conditions in Proposition 2.19 only concern folded assignments, $\mathcal{G}'$ satisfies these conditions. □

This means that sampling from $\mathcal{G}$ can be written as $(x, y) \sim \mathcal{G}$, meaning that we have sampled the constraint "$x = y$".

2.3 The Had_k-to-2-Lin(2) Gadget

Now we focus on our main setting, which is constructing a Had_k-to-2-Lin(2) gadget. Via Section 2.2, we need only consider how well the gadget does against folded assignments.

The Had_k predicate has $2^k - 1$ variables. In addition, it has $K := 2^k$ satisfying assignments, one for each setting of the variables $x_{\{1\}}$ through $x_{\{k\}}$. It will often be convenient to take an alternative (but equivalent) viewpoint of the variable set $V := \{-1, 1\}^K$ as the set of k-variable Boolean functions, i.e.

$$V = \left\{ f \mid f : \{-1, 1\}^k \to \{-1, 1\} \right\}.$$

The Had_k matrix is a $2^k \times (2^k - 1)$ matrix whose rows are indexed by strings in $\{-1, 1\}^k$ and whose columns are indexed by nonempty subsets $S \subseteq [k]$. The (x, S)-entry of this matrix is $\chi_S(x)$. This can be verified by noting that for any $x \in \{-1, 1\}^k$,

$$\big(\chi_{\{1\}}(x), \chi_{\{2\}}(x), \ldots, \chi_{\{k\}}(x), \chi_{\{1,2\}}(x), \ldots, \chi_{\{1,2,\ldots,k\}}(x),\big)$$

is a satisfying assignment of the Had_k predicate. As a result, for each $S \neq \emptyset$, χ_S is a column in the Had_k matrix. Therefore, these functions are the generic primary variables. However, it will be convenient to consider a larger set of functions to be primary. For example, because we plan on using our gadget on folded assignments, χ_S and $-\chi_S$ will always have opposite values, and so the $-\chi_S$'s should also be primary variables. In addition, it is a little unnatural to have every parity function but one be a primary variable, so we will include the constant function $\chi_\emptyset$ and its negation $-\chi_\emptyset$ in the set of primary variables. In total, we have the following definition.

Definition 2.21. The *primary variables* of a Had_k-to-2-Lin(2) gadget are the functions $\pm\chi_S$, for any $S \subseteq [k]$. The remaining functions are auxiliary variables.

To account for the inclusion of $\chi_\emptyset$ as a primary variable, we will have to modify some of our definitions from Section 2.1. We begin by defining a modification to the Had_k predicate.

Definition 2.22. The Had_k^* predicate has 2^k input variables, one for each subset $S \subseteq [k]$. The input string $\{x_S\}_{S\subseteq[k]}$ satisfies Had_k^* if for each S, $x_S = x_\emptyset \cdot \prod_{i\in S} x_{\{i\}}$.

In other words, if $x_\emptyset = 1$, then the remaining variables should satisfy the Had_k predicate, and if $x_\emptyset = -1$, then their negations should. We will say that A *satisfies the* Had_k^* *predicate* if

$$\mathsf{Had}_k^*\left(A\left(\chi_\emptyset\right), A\left(\chi_{\{1\}}\right), \ldots, A\left(\chi_{\{k\}}\right), A\left(\chi_{\{1,2\}}\right), \ldots, A\left(\chi_{[k]}\right)\right) = 1.$$

Otherwise, A *fails to satisfy the* Had_k^* *predicate.* We say that $\mathcal{A}$ is *random on the primary variables* if the tuple

$$\left(A\left(\chi_\emptyset\right), A\left(\chi_{\{1\}}\right), \ldots, A\left(\chi_{\{k\}}\right), A\left(\chi_{\{1,2\}}\right), \ldots, A\left(\chi_{[k]}\right)\right)$$

is, over a random $A \sim \mathcal{A}$, distributed as a uniformly random element of $\{-1,1\}^K$.

Definition 2.23. Denote by $\mathsf{R}(\mathsf{Had}_k)$ the set of folded distributions which are uniformly random on the primary variables.

Definition 2.24. A (c,s)-*gadget* reducing Max-Had_k to Max-2-Lin(2) is a gadget $\mathcal{G}$ satisfying the following properties:

- (Completeness): For every dictator assignment d_i, $\mathsf{uval}(d_i;\mathcal{G}) \le c$.
- (Soundness): For any $\mathcal{A} \in \mathsf{R}(\mathsf{Had}_k)$, $\mathsf{uval}(\mathcal{A};\mathcal{G}) \ge s$.

Proposition 2.25. *The following two statements are equivalent:*

1. *There exists a* (c,s)*-gadget reducing Max-*Had_k *to Max-*2-Lin(2)*.*
2. *There exists a* (c,s)*-generic gadget reducing Max-*Had_k *to Max-*2-Lin(2)*.*

Proof. We prove the two directions separately.

(1) ⇒ (2): Let $\mathcal{G}$ be a (c,s)-gadget reducing Max-Had_k to Max-2-Lin(2). We claim that for any folded $\mathcal{A} \in \mathsf{R}^{gen}(\mathsf{Had}_k)$, $\mathsf{uval}(\mathcal{A};\mathcal{G}) \ge s$. To see this, consider the distribution $\mathcal{A}' \in \mathsf{R}(\mathsf{Had}_k)$ which samples $A \sim \mathcal{A}$ and outputs either A or $-A$, each with half probability. Then $\mathsf{uval}(\mathcal{A}';\mathcal{G}) = \mathsf{uval}(\mathcal{A};\mathcal{G})$, and furthermore we know that $\mathsf{uval}(\mathcal{A};\mathcal{G}) \ge s$. As a result, $\mathcal{G}$ satisfies the (Completeness) and (Soundness) conditions in the statement of Proposition 2.19, meaning there exists a (c,s)-generic gadget reducing Max-Had_k to Max-2-Lin(2).

(2) ⇒ (1): Let $\mathcal{G}$ be a (c,s)-*generic* gadget reducing Max-Had_k to Max-2-Lin(2). Let $\mathcal{A} \in \mathsf{R}(\mathsf{Had}_k)$, and for $b \in \{-1,1\}$, write $\mathcal{A}_{(b)}$ for $\mathcal{A}$ conditioned on the variable $\chi_\emptyset$ being assigned the value b. Then $b \cdot \mathcal{A}_{(b)}$ (by which we mean the distribution where we sample $A \sim \mathcal{A}_{(b)}$ and output $b \cdot A$) is in $\mathsf{R}^{gen}(\mathsf{Had}_k)$ for both $b \in \{-1,1\}$, and so $\mathsf{uval}\left(b \cdot \mathcal{A}_{(b)};\mathcal{G}\right) \ge s$. As $\mathsf{uval}(\mathcal{A}_{(b)};\mathcal{G}) = \mathsf{uval}\left(b \cdot \mathcal{A}_{(b)};\mathcal{G}\right)$, $\mathsf{uval}(\mathcal{A};\mathcal{G}) \ge s$, and so $\mathcal{G}$ is a (c,s)-gadget reducing Max-Had_k to Max-2-Lin(2). □

Combining this with Proposition 2.17, we have the following corollary.

Corollary 2.26. *Suppose there exists a (c, s)-gadget reducing Max-Had_k to Max-2-Lin(2). Then for all $\epsilon > 0$, given an instance $\mathcal{I}$ of Max-2-Lin(2), it is NP-hard to distinguish between the following two cases:*

- *(Completeness):* $\mathsf{uval}(\mathcal{I}) \leq c + \epsilon$.
- *(Soundness):* $\mathsf{uval}(\mathcal{I}) \geq s - \epsilon$.

2.4 Reducing to the length-one case

When constructing good gadgets, we generally want dictators to pass with as high of probability as possible. By Proposition 2.20, we can assume that our gadget operates by sampling an edge (x, y) and testing equality between the two endpoints. Any such edge of Hamming distance i will be violated by $\frac{i}{K}$ of the dictator assignments. Intuitively, then, if we want to dictators to pass with high probability, we should concentrate the probability mass of our gadget $\mathcal{G}$ on edges of low Hamming distance. The following proposition shows that this is true in the extreme: so long as we are only concerned with maximizing the quantity $\frac{s}{c}$, we can always assume that $\mathcal{G}$ is entirely supported on edges of Hamming distance one.

Proposition 2.27. *Suppose there exists a (c, s)-gadget $\mathcal{G}$ reducing Max-Had_k to Max-2-Lin(2). Then there exists a (c', s')-gadget reducing Max-Had_k to Max-2-Lin(2) using only length-one edges for which*

$$\frac{s'}{c'} \geq \frac{s}{c}.$$

Proof. For each $i \in \{1, \ldots, K\}$, let p_i be the probability that an edge sampled from $\mathcal{G}$ has length i, and let $\mathcal{G}_i$ denote the distribution of $\mathcal{G}$ conditioned on this event. Then sampling from $\mathcal{G}$ is equivalent to first sampling a length i with probability p_i, and then sampling an edge from $\mathcal{G}_i$.

Let $Q = 1 \cdot p_1 + 2 \cdot p_2 + \ldots + K \cdot p_K$, and for each $i \in \{1, \ldots, K\}$ define $q_i = \frac{i \cdot p_i}{Q}$. It is easy to see that the q_i's form a probability distribution. Now we may define the new gadget $\mathcal{G}'$ as follows:

1. Sample a length i with probability q_i.
2. Sample $(x, y) \sim \mathcal{G}_i$.
3. Pick an arbitrary shortest path $x = x_0, x_1, \ldots, x_i = y$ through the hypercube $\{-1, 1\}^K$.
4. Output a uniformly random edge (x_j, x_{j+1}) from this path.

Note that $\mathcal{G}'$ only uses length-one edges. Let $\mathcal{G}'_i$ denote the distribution of $\mathcal{G}'$ conditioned on i being sampled in the first step. (Note that $\mathcal{G}'_i$ is defined in a way that is different from the way $\mathcal{G}_i$ is defined.)

Let $A : \{-1, 1\}^K \to \{-1, 1\}$ be any assignment. Then

$$\mathsf{uval}(A; \mathcal{G}) = \sum_{i=1}^{K} p_i \cdot \mathsf{uval}(A; \mathcal{G}_i), \qquad \text{and} \qquad \mathsf{uval}(A; \mathcal{G}') = \sum_{i=1}^{K} q_i \cdot \mathsf{uval}(A; \mathcal{G}'_i).$$

We can relate $\mathsf{uval}(A; \mathcal{G}'_i)$ to $\mathsf{uval}(A; \mathcal{G}_i)$ as follows: if A assigns different values to the endpoints of the edge $(x, y) \sim \mathcal{G}$, then on any shortest path $x = x_0, x_1, \ldots, x_i = y$ through the hypercube $\{-1, 1\}^K$, A must assign different values to at least one of the edges (x_j, x_{j+1}). As a result, every time A errs on $\mathcal{G}_i$, it must err at least a $(1/i)$-fraction of the time on $\mathcal{G}'_i$. This means that:

$$\mathsf{uval}(A; \mathcal{G}'_i) \geq \frac{\mathsf{uval}(A; \mathcal{G}_i)}{i}. \tag{1}$$

In the case when A is a dictator function, Equation (1) becomes an equality. This is because $x = x_0, x_1, \ldots, x_i = y$ is a shortest path between x and y through the hypercube $\{-1,1\}^K$. If A assigns the same values to x and y, then it will assign the same values to all of $x_0, x_1, \ldots, x_i$. If, on the other hand, it assigns different values to x and y, then it will assign different values to the endpoints of exactly one edge (x_j, x_{j+1}).

Now we can use this relate $\mathsf{uval}(A;\mathcal{G}')$ to $\mathsf{uval}(A;\mathcal{G})$:

$$\begin{aligned}\mathsf{uval}(A;\mathcal{G}') &= \sum_{i=1}^{K} q_i \cdot \mathsf{uval}(A;\mathcal{G}_i') \\ &\geq \sum_{i=1}^{K} \left(\frac{i \cdot p_i}{Q}\right) \cdot \frac{\mathsf{uval}(A;\mathcal{G}_i)}{i} \\ &= \frac{1}{Q}\sum_{i=1}^{K} p_i \cdot \mathsf{uval}(A;\mathcal{G}_i) \\ &= \frac{1}{Q}\mathsf{uval}(A;\mathcal{G}). \end{aligned} \tag{2}$$

Here the inequality follows from the definition of q_i and Equation (1). As Equation (1) is an equality in the case when A is a dictator function, we have that $\mathsf{uval}(A;\mathcal{G}') = \frac{1}{Q}\mathsf{uval}(A;\mathcal{G})$ in this case.

Let $\mathcal{A} \in \mathsf{R}(\mathsf{Had}_k)$ maximize $\mathsf{val}(\mathcal{A};\mathcal{G}')$, and let d_i be any dictator function. Then

$$\frac{s'}{c'} = \frac{\mathsf{uval}(\mathcal{A};\mathcal{G}')}{\mathsf{uval}(d_i;\mathcal{G}')} \geq \frac{\frac{1}{Q}\mathsf{uval}(\mathcal{A};\mathcal{G})}{\frac{1}{Q}\mathsf{uval}(d_i;\mathcal{G})} = \frac{\mathsf{uval}(\mathcal{A};\mathcal{G})}{\mathsf{uval}(d_i;\mathcal{G})} \geq \frac{s}{c}.$$

Here the first inequality is by Equation (2) (and the fact that it is an equality for dictators), and the second inequality follows from the fact that $\mathsf{uval}(\mathcal{A},\mathcal{G}) \geq s$ and $\mathsf{uval}(d_i,\mathcal{G}) = c$. □

2.5 Linear programs

One of the key insights of the paper [TSSW00] is that optimal gadgets (as per Definition 2.14) can be computed by simply solving a linear program. Fortunately, the same holds for computing optimal gadgets as per Definition 2.24. In our case, the appropriate linear program (taking into account Proposition 2.27) is:

$$\begin{aligned} \max \quad & s \\ \text{s.t.} \quad & \mathsf{uval}(\mathcal{A};\mathcal{G}) \geq s, \quad \forall \mathcal{A} \in \mathsf{R}(\mathsf{Had}_k), \\ & \mathcal{G} \text{ is a gadget supported on edges of length one.} \end{aligned}$$

As written, this linear program has an (uncountably) infinite number of constraints, but this can fixed by suitably discretizing the set $\mathsf{R}(\mathsf{Had}_k)$. This is not so important for us, as even after performing this step, the linear program is simply too large to ever be feasible in practice. What is important for us is that we can take its dual; doing so yields the following linear program:

Definition 2.28. The *dual LP* is defined as

$$\begin{aligned} \min \quad & s \\ \text{s.t.} \quad & \mathop{\mathbf{Pr}}_{A\sim\mathcal{A}}[A(x) = A(y)] \leq s, \quad \forall \text{ edges } (x,y) \text{ of length one}, \\ & \mathcal{A} \in \mathsf{R}(\mathsf{Had}_k). \end{aligned}$$

The dual linear program shows us that we can upper-bound the soundness of any gadget with the value s by exhibiting a distribution on assignments in $\mathsf{R}(\mathsf{Had}_k)$ which passes each length-one edge with probability at least s. Moreover, strong LP duality tells us that the optimum values of the two LPs are the same. Hence, we can prove a *tight* upper bound by exhibiting the right distribution. We do this in Section 4 for gadgets reducing Max-Had_3 to Max-2-Lin(2).

2.6 The Had_3 gadget

In this section, we will prove some structural results about the hypercube $\{-1,1\}^8$ which are relevant to any Had_3-to-2-Lin(2) gadget. The results of this section will be useful for both Sections 3 and 4.

Given a string $x \in \{-1,1\}^n$ and subset of strings $B \subseteq \{-1,1\}^n$, we define the distance of x to B as $d_H(x,B) := \min_{y \in B} d_H(x,y)$.

Proposition 2.29. *The vertex set $V = \{-1,1\}^8$ can be partitioned as $V = V_0 \cup V_1 \cup V_2$, in which V_0 is the set of primary variables, and $V_i = \{x \in V \mid d_H(x,V_0) = i\}$, for $i = 1,2$.*

Proposition 2.30. $|V_0| = 16, \quad |V_1| = 128, \text{ and } |V_2| = 112.$

Proposition 2.31.

- *Each $x \in V_0$ has eight neighbors in V_1.*
- *Each $x \in V_1$ has one neighbor in V_0 and seven neighbors in V_2.*
- *Each $x \in V_2$ has eight neighbors in V_1. Furthermore, there are four elements of V_0 which are Hamming distance two away from x.*

Proposition 2.32. *Let $f \in V_2$, and let g_1, g_2, g_3, and g_4 be the four elements of V_0 which are Hamming distance two away from f. Then for any $x \in \{-1,1\}^3$, three of the g_i's have the same value and one has a different value, and $f(x) = \mathsf{sign}(g_1(x) + g_2(x) + g_3(x) + g_4(x))$.*

Proof of Propositions 2.29, 2.30, 2.31, and 2.32. In this proof, we will take the viewpoint of V as the set of 3-variable Boolean functions. The primary variables are of the form $\pm\chi_S$, where $S \subseteq [3]$. There are 16 such functions, and so $|V_0| = 16$.

Let f' differ from one of the primary variables on a single input. From above, it must be at least distance 3 from any of the other primary variables. This immediately implies that f''s seven other neighbors are in V_2. There are $16 \cdot 8 = 128$ distinct ways of constructng f', and so $|V_1| = 128$.

This leaves $256 - 16 - 128 = 112$ variables in V not yet accounted for. We will now show a method for constructing 112 different elements of V_2; by the pigeonhole principle, this shows that V can be partitioned as Proposition 2.29 guarantees. Given three primary variables $b_1\chi_{S_1}$, $b_2\chi_{S_2}$, and $b_3\chi_{S_3}$, where $b_1, b_2, b_3 \in \{-1,1\}$, set $b_4 := -b_1 \cdot b_2 \cdot b_3$ and $S_4 := S_1 \Delta S_2 \Delta S_3$. Consider the function $f : \{-1,1\}^3 \to \{-1,1\}$ defined as

$$f(x) := \mathsf{sign}\left(b_1\chi_{S_1}(x) + b_2\chi_{S_2}(x) + b_3\chi_{S_3}(x) + b_4\chi_{S_4}(x)\right).$$

Our claim is that f is distance-2 from each of the $b_i\chi_{S_i}$'s. First, to see that this $\mathsf{sign}(\cdot)$ is well-defined, note that by definition, $\prod_{i=1}^4 b_i\chi_{S_i}(x) = -1$ for all $x \in \{-1,1\}^3$. As a result, for any x, three of the $b_i\chi_{S_i}(x)$'s have the same value, while the other one has a different value. This means that

$$\sum_{i=1}^4 b_i\chi_{S_i}(x) = 2 \cdot \mathsf{sign}\left(\sum_{i=1}^4 b_i\chi_{S_i}(x)\right).$$

for all x. Thus, the correlation of any of the $b_i\chi_{S_i}$'s with f is

$$\mathop{\mathbf{E}}_{\boldsymbol{x}}[f(\boldsymbol{x}) \cdot b_i\chi_{S_i}] = \frac{1}{2}\mathop{\mathbf{E}}_{\boldsymbol{x}}\left[\left(\sum_{i=1}^{4} b_i\chi_{S_i}(x)\right) \cdot b_i\chi_{S_i}\right] = \frac{1}{2}.$$

In other words, $\mathbf{Pr}_{\boldsymbol{x}}[f(\boldsymbol{x}) = b_i\chi_{S_i}] = \frac{3}{4}$ for each $i \in \{1, \ldots, 4\}$.

There are 8 neighbors of f; each $b_i\chi_{S_i}$ neighbors two of them. As a result, all of f's neighbors are in V_1. In addition, since they are neighbors to the $b_i\chi_{S_i}$'s, they can't be neighbors for any of the other primary variables. This means that the only variables in V_0 that f is distance 2 from are the $b_i\chi_{S_i}$'s.

There are $2 \cdot \binom{8}{3} = 112$ ways of selecting the $b_i\chi_{S_i}$'s. As there are only 112 variables in V which are not in either V_0 or V_1, all of the remaining variables in V must be contained in V_2, and they must all be generated in the manner above. □

Proposition 2.33. *Let $B = \mathsf{sat}(\mathsf{Had}_3^*)$. Then*

$$\mathop{\mathbf{Pr}}_{\boldsymbol{x}}[d_H(\boldsymbol{x}, B) = 0] = \frac{1}{16}, \quad \mathop{\mathbf{Pr}}_{\boldsymbol{x}}[d_H(\boldsymbol{x}, B) = 1] = \frac{1}{2}, \text{ and } \mathop{\mathbf{Pr}}_{\boldsymbol{x}}[d_H(\boldsymbol{x}, B) = 2] = \frac{7}{16},$$

where $\boldsymbol{x}$ is a uniformly random element of $\{-1, 1\}^8$.

Proof. This can be proven using a proof similar to Proposition 2.30. Alternatively, we can just show a direct correspondence between the setting here and the setting in Proposition 2.30, as follows.

The input to Had_3^* is a set of bits $\{x_S\}_{S\subseteq[k]}$, which can also be thought of as the function $f : \mathcal{P}(\{1, 2, 3\}) \to \{-1, 1\}$ in which $f(S) := x_S$. The satisfying assignments are then any function of the form $S \to b \cdot \chi_S(x)$, where $b \in \{-1, 1\}$ and $x \in \{-1, 1\}^3$ are both fixed. For a string $x \in \{-1, 1\}^3$, let $\alpha(x)$ be the corresponding set, i.e. $\alpha(S)_i = -1$ if and only if $i \in S$. For any function $f : \mathcal{P}(\{1, 2, 3\}) \to \{-1, 1\}$, we can associate it with the function $\alpha(f) : \{-1, 1\}^3 \to \{-1, 1\}$ defined by $\alpha(f)(x) := f(\alpha(x))$ for all x. Then α maps any satisfying assignment to Had_3^* into one of the primary variables in V_0, and more generally, $d_H(f, B) = i$ if and only if $\alpha(f) \in V_i$. The proposition therefore follows by applying Proposition 2.30 and by noting that $\frac{16}{256} = \frac{1}{16}$, $\frac{128}{256} = \frac{1}{2}$, and $\frac{112}{256} = \frac{7}{16}$. □

Proposition 2.34.

1. *Let $f, g \in V_0$ be a pair of distinct affine functions. Then either $d_H(f, g) = 8$, or $d_H(f, g) = 4$.*
2. *For any $x, y \in \{-1, 1\}^3$, $x \neq y$, $b_x, b_y \in \{-1, 1\}$, the number of functions $f \in V_0$ such that $f(x) = b_x$ is 8, and the number of functions $f \in V_0$ such that $f(x) = b_x$ and $f(y) = b_y$ is 4.*

Proof. Proof of (1): Let $f = b_f\chi_S$, and $g = b_g\chi_T$. Then $\mathbf{E}[fg] = b_f b_g \mathbf{E}[\chi_{S\Delta T}]$ where Δ is the symmetric difference of two sets. If $f = -g$, then clearly $d_H(f, g) = 8$. Now we assume that $f \neq \pm g$, and therefore $S \neq T$. Then $\mathbf{E}[\chi_{S\Delta T}] = 0$. This completes the proof.

Proof of (2): Consider function $f(x) = a_0 + a_1x_1 + a_2x_2 + a_3x_3$. Construct a linear system where a_0, a_1, a_2, a_3 are the variables and $f(x) = b_x$ and $f(y) = b_y$ are the constraints. The result follows from working out the size of the solution space. □

2.7 Reducing to Max-Cut

Definition 2.35. Let $(\neq) : \{-1,1\}^2 \to \{0,1\}$ be the inequality predicate, i.e. $(\neq)(b_1,b_2) = 1$ iff $b_1 \neq b_2$ for all $b_1, b_2 \in \{-1,1\}$. The Max-Cut CSP is the special case of the Max-$(\neq)$ CSP in which every constraint $\mathcal{C} = ((x_1,b_1),(x_2,b_2))$ satisfies $b_1 = b_2 = 1$. In other words, every constraint is of the form "$x_1 \neq x_2$".

Proposition 2.36. *For some predicate ϕ, suppose $\mathcal{G}$ is (c,s)-generic gadget reducing Max-ϕ to Max-*2-Lin(2)*. Then there exists a (c',s')-gadget reducing Max-ϕ to* Max-Cut *satisfying*

$$\frac{s'}{c'} = \frac{s}{c}.$$

Proof. Suppose the vertex set of $\mathcal{G}$ is $V = \{-1,1\}^K$. Let $\mathcal{G}'$ be the gadget which operates as follows:

1. With probability $1 - \frac{1}{2^{K-1}}$, pick $x \in \{-1,1\}^K$ and output the constraint "$x \neq -x$".
2. Otherwise, sample $\mathcal{C} \sim \mathcal{G}$. If $\mathcal{C}$ is of the form "$x \neq y$", output "$x \neq y$". If $\mathcal{C}'$ is of the form "$x = y$", output "$x \neq -y$".

Any folded assignment A fails $\mathcal{G}$' with probability at most $\frac{1}{2^{K-1}}$. Any assignment A which is *not* folded fails $\mathcal{G}$' with probability at least $\frac{1}{2^{K-1}}$. As a result, we can always assume that any assignment is folded.

Now, if A is folded, then for any $x, y \in \{-1,1\}^K$, $A(x) = A(y)$ if and only if $A(x) \neq A(-y)$. As a result, $\mathsf{uval}(A;\mathcal{G}') = \mathsf{uval}(A;\mathcal{G})/2^{k-1}$. Thus, $c' = c/2^{k-1}$, $s' = s/2^{k-1}$, and so $s'/c' = s/c$. □

3 The factor-11/8 hardness result

In this section, we prove the following theorem.

Theorem 3.1. *There is a $\left(\frac{1}{8}, \frac{11}{64}\right)$-gadget reducing Had_3 to* 2-Lin(2).

Using Propositions 2.17 and 2.36, we have the following two corollaries:

Corollary 3.2. *There is a (c,s)-generic gadget reducing Had_3 to* Max-Cut *with $\frac{s}{c} = \frac{11}{8}$.*

Corollary 3.3 (Theorem 1.7 restated)**.** *Fix any $C < \frac{11}{8}$. Then it is NP-hard to achieve a factor-C approximation for both the Max-*2-Lin(2) *and the* Max-Cut *CSPs.*

Proof of Theorem 3.1. To construct our gadget, we will assign a nonnegative weight to each edge in the gadget. Our gadget will then sample each edge with probability equal to its weight normalized by the weight of the entire gadget. As argued in Proposition 2.27, the gadget will only use length-one edges. For $f, g \in V$ with $d_H(f,g) = 1$, the weight of the edge $\{f,g\}$ is 5 if and only if either $f \in V_0$ or $g \in V_0$, and otherwise the weight is 1. The total weight of the edges in $\mathcal{G}$ is $5 \times 128 + 896 = 1536$.

For the completeness, the fact that the dictators pass with probability $\frac{7}{8}$ follows immediately from the fact that $\mathcal{G}$ only uses edges of length one. For the soundness, let $\mathcal{A} \in \mathsf{R}(\mathsf{Had}_3)$. We will lower-bound $\mathsf{uval}(\mathcal{A};\mathcal{G})$ by upper bounding $\mathsf{uval}(A;\mathcal{G})$ for each A in the support of $\mathcal{A}$ in terms of how close A's assignment to the primary variables is to satisfying the Had_3^* predicate.

Lemma 3.4. *Let $A : \{-1,1\}^8 \to \{-1,1\}$. If A's assignment to the primary variables satisfies the Had_3^* predicate, then $\mathsf{uval}(A;\mathcal{G}) \geq \frac{1}{8}$.*

Lemma 3.5. *Let $A : \{-1,1\}^8 \to \{-1,1\}$. If A's assignment to the primary variables is distance one from satisfying the Had_3^* predicate, then $\mathsf{uval}(A;\mathcal{G}) \geq \frac{21}{128}$.*

Lemma 3.6. *Let $A : \{-1,1\}^8 \to \{-1,1\}$. If A's assignment to the primary variables is distance two from satisfying the Had_3^* predicate, then $\mathsf{uval}(A;\mathcal{G}) \geq \frac{3}{16}$.*

Proposition 2.33 gives the probability that a random $A \sim \mathcal{A}$ will fall into each of these three cases. In total

$$\mathsf{uval}(\mathcal{A};\mathcal{G}) \geq \frac{1}{16}\cdot\frac{1}{8} + \frac{1}{2}\cdot\frac{21}{128} + \frac{7}{16}\cdot\frac{3}{16} = \frac{11}{64},$$

which is what the theorem guarantees.

Before proving these lemmas, we will do some preliminary work which is relevant to all three. In the remaining part of this section, we fix a partial assignment A that assigns values to variables in V_0. To analyze the quality of the gadget, we analyze certain measures of the gadget and bound the best possible way to complete A to a full assignment. Hence, all definitions in the rest of this section will be with respect to the partial assignment A.

We classify variables in V_2 by the assignments of their associated affine functions.

Definition 3.7. Let $f \in V_2$ be a function associated with $g_0, g_1, g_2, g_3 \in V_0$. We say that f is a $(4,0)$ function if $A(g_0) = A(g_1) = A(g_2) = A(g_3)$. Similarly, we define $(3,1)$ and $(2,2)$ functions.

We consider paths of length 2 that start at some $f \in V_2$ and ends at one of the affine functions $g_i \in V_0$.

Definition 3.8. A path of length 2 from $f \in V_2$ to $g \in V_0$ is good if $A(f) = A(g)$. Otherwise it is a bad path. Let B be the number of bad paths given by partial assignment A.

Given partial assignment A, an easy way to assign values to some of the variables in V_2 is to take the majority value of its associated affine functions if the function is of type $(4,0)$ or $(3,1)$, and leave it undetermined if it is of type $(2,2)$. This gives us the following way of classifying functions in V_1.

Definition 3.9. Consider a function $f \in V_1$ and a partial assignment to variables in $V_0 \cup V_2$. We say that f is of type (a,b,c) if in the partial assignment, there are a good paths, b bad paths and c undetermined-paths going through f. Note that $a+b+c = 7$.

A function is *switched* if it has type $(0,7,0)$.

After fixing the assignment to the $(4,0)$ and $(3,1)$ functions, the maximum number of switched functions is the number of functions in V_1 that have so far no good paths through them.

Let C be the number of switched functions.

Once we have assignment for variables in $V_0 \cup V_2$, we can find the optimal assignments for variables in V_1 in a greedy way. That is, for an (a,b,c) function in $f \in V_1$, as long as $b < 7$, we set $A(f) = A(g)$ where $g \in V_0$ is the closest affine function to f, and otherwise we set $A(f) = -A(g)$. The weight of violated edges of this assignment is $B - 2C$.

The overall proof idea is to show that the above majority assignment is always the best. In particular, we prove that no matter how we change the assignment of some of the $(4,0)$ and $(3,1)$ variables to anti-majority and increase the number of bad paths B by some number Q, we will never be able to increase the number of functions with no good paths through them by more than $Q/2$. Thus, the weight of violated constraints will never decrease under non-majority assignments.

Once we fix the assignment of $(4,0)$ and $(3,1)$ to majority, it is not hard to compute the number of bad paths and the maximum number of switched function in each of the two cases. In particular,

in the distance 1 case, the minimum weight of violated constraints is 252 ($= \frac{21}{128} \cdot 1536$), and the distance 2 case has minimum weight at least 288 ($= \frac{3}{16} \cdot 1536$).

This completes the proof. □

3.1 Assignments at distance 1 from Had_3

In this section, we prove Lemma 3.5.

Proof of Lemma 3.5: Let A be a partial assignment to variables in V_0, where there exists $x_0 \in \{-1,1\}^3$, and $l_0 \in V_0$, such that $A(f) = f(x_0)$ for all $f \in V_0 \setminus \{l_0, -l_0\}$, and $A(b \cdot l_0) = -b \cdot l_0(x_0)$ for $b \in \{-1,1\}$. We call $\pm l_0$ the corrupted affine functions. As the gadget has a total weight of 1536, we want to show that the total weight violated by A is at least 252.

Of the 112 functions in V_2, 56 of them are associated with $\pm l_0$, and 56 of them are not. The 56 functions that are not associated with $\pm l_0$ are all $(3,1)$ functions by Proposition 2.32.

Let $f \in V_2$ be a function that is associated with $a_0 \in \{l_0, -l_0\}$. There are 14 of these functions such that $a_0(x_0) \neq f(x_0)$, and in this case f is a $(4,0)$ function. Otherwise it is a $(2,2)$ function.

As discussed above, given a partial assignment to V_0, we need to decide, for the $(4,0)$ and $(3,1)$ functions in V_2, whether we assign them the majority assignment. The analysis in this section proceed in two steps. We first argue that for any assignment for the $(3,1)$ variables, to minimize the weight of violated constraints, either all $(4,0)$ variables are assigned according to majority of their associated functions in V_0, or all of them are assigned according to anti-majority. It is easy to argue that the cost will be high in the case where all $(4,0)$ variables are assigned according to anti-majority. Then, assuming that the $(4,0)$ variables are assigned according to majority, we prove that the $(3,1)$ variables should also be assigned according to majority. This gives us a sufficient lower-bound for the weight of violated constraints.

Before we argue about the optimal assignments, let us classify the variables with respect to the majority assignment for V_2 and see how different classes of variables relate to each other.

Proposition 3.10.

1. *There are 2 variables of type* $(7,0,0)$*. Those are obtained by starting from* $\pm l_0$ *and flipping the value at* x_0*. All 7 neighbors in* V_2 *of the* $(7,0,0)$ *functions have type* $(4,0)$*. This gives a total of 14, and those are exactly all the* $(4,0)$ *functions.*

2. *Each* $(4,0)$ *function has 1 neighbor of type* $(7,0,0)$*, 1 of type* $(1,0,6)$*, and 6 of type* $(6,0,1)$*.*

3. *Each* $(6,0,1)$ *function that is adjacent to a* $(4,0)$ *function actually has 2 neighbors of type* $(4,0)$*, 1 of type* $(2,2)$ *and 4 of type* $(3,1)$*. There are 42 such* $(6,0,1)$ *functions.*

Proof. Consider a path starting from $a_0 \in V_0$, where $a_0 = \pm l_0$. Let f_1 be the function obtained by flipping the value of a_0 at x_0. We then flip some other values to get to function $f \in V_2$. Since $f(x_0) \neq a_0(x_0)$, we know that a_0 is the unique function that has different value at x_0 from f in Proposition 2.32, and now since $A(f) = -a_0(x_0)$, we have that f is a $(4,0)$ variable. There are $2 \times 7 = 14$ such paths, and each of them end at a distinct $(4,0)$ variable. This argument also shows that all 7 V_2-neighbors of f_1 are of $(4,0)$-type and hence such f_1 is a $(7,0,0)$ variable.

We now study the other neighbors of the $(4,0)$ variables. Starting from such an f, if we flip the value at x_0, then we arrive at function f_1' associated with a_0 and they agree on the value of x_0 but differ on exactly one point, denoted as y_0. If we flip the value at a point that is neither x_0 nor x_0', then we arrive at another function in V_2 which is of $(2,2)$-type. This means that f_1' is a $(1,0,6)$ function.

To understand the other neighbors of f, note that by Proposition 2.34, there are 4 affine functions that takes values different from a_0 at both x_0 and y_0, one of them is $-a_0$, and the remaining three are all at distance 2 from f. To each of those affine functions there are 2 paths from f so this gives a total of 6 paths. We now show that all six V_1 variables on these 6 paths have type $(6, 0, 1)$.

Let $f_2 \in V_1$ be one of the functions on these 6 paths, and let a_1 be its associated affine function. We know that $a_0(x_0) = f_2(x_0) \neq a_0(x_0)$. Now consider the affect of flipping different values in f_2.

- To get to the closest $(7, 0, 0)$ function f_1 from f_2, we need to flip two bits. We can flip them in two different orders, so this gives us two different paths of length 2, each going through a different $(4, 0)$ function.
- If we flip value x_0 of function f_2 and get function f_3, then in fact we have that f_3 is still distance 2 from a_0 and thus associated with it. Note that $a_1(x_0) \neq f_3(x_0) = a_0(x_0)$, which means that $A(a_1) = A(a_0) \neq f_3(x_0)$, and thus f_3 is a $(2, 2)$ function.
- For all the other neighbors of f_2, we have that they agree with f_2 on x_0 and is at distance 4 from a_0. This means that those functions are not associated with $\pm l_0$, and therefore are $(3, 1)$ functions contributing a good path to f_2.

This concludes the analysis. □

We also need to analyze the remaining variables in V_1 and the neighbors of the $(3, 1)$ variables.

Proposition 3.11.

- *There are 56 functions in V_1 of type $(3, 1, 3)$, and 14 in V_1 of type $(0, 4, 3)$. These functions do not have neighbors of type $(4, 0)$.*
- *For $(3, 1)$ functions in V_2, they contribute 3 good paths to 3 $(3, 1, 3)$ functions, 3 good paths to 3 $(6, 0, 1)$ functions, and 2 bad paths to 2 $(0, 4, 3)$ functions.*

Proof. Let a_1 be an arbitrary affine function that is not corrupted, and a_0 now be the corrupted affine function such that $a_0(x_0) = a_1(x_0)$. Choose one of the four bits where a_0 and a_1 differ, let's assume that we have chosen y_0.

Starting from a_1, let g_1 be the function where we flip the value $a_1(y_0)$. If we flip any of the three remaining bits where g_1 and a_0 differ, we will end up in a $(2, 2)$ function. Otherwise, we will end up in a $(3, 1)$ function. There are two subcases here. If we flip x_0 and get function g_2, then taking the majority assignment for g_2 will give us a bad path at g_1. In the other three cases, we will get a good path. The conclusion is that in this case g_1 is a $(3, 1, 3)$ function. There are $14 \times 4 = 56$ such functions.

Now consider what happens if we start at a_1 and flip the value at x_0 first. If we then flip any of the three remaining bits where a_0 and a_1 agree, then we end up in a $(2, 2)$ function. Otherwise, we end up in a $(3, 1)$ function that contributes a bad path. This gives us 14 level 1 functions of type $(0, 4, 3)$.

Together with Proposition 3.10, we see that there are 2 $(7, 0, 0)$ functions, 56 $(3, 1, 3)$ functions, 14 $(0, 4, 3)$ functions, 42 $(6, 0, 1)$ functions, and 14 $(1, 0, 6)$ functions. This gives the types of all 128 functions in V_1.

We now view it from the perspective of $(3, 1)$ functions. For any such function $f_2 \in V_1$, there are 6 good paths coming from it and going into 3 uncorrupted V_0 variables, and 2 bad paths going to another uncorrupted variable in V_0. Consider 2 good paths that go to the same affine function

a_1. From the above analysis, we know that the level 1 functions on them have type either $(3,1,3)$ or $(6,0,1)$. Let a_0 be the corrupted affine function such that $a_1(x_0) = a_0(x_0)$. Combining the above analysis, we know that on these 2 paths, we flipped exactly one bit x such that $a_1(x) = a_0(x)$ and another bit y such that $a_1(y) \neq a_0(y)$, and $x \neq x_0$. We also know that starting from a_1, if we first flip y, we will get a $(3,1,3)$ function. If we first flip x, we will actually get a $(6,0,1)$ function because it has a $(4,0)$-neighbor and it is clearly not a $(7,0,0)$ function.

Therefore each of the $(3,1)$ functions contributes to 3 $(3,1,3)$ functions exactly 1 good path each, and 3 $(6,0,1)$ functions exactly 1 good path each. □

The following proposition show that in an optimal assignment, all $(4,0)$ variables should be assigned according to majority, unless all of them are flipped.

Proposition 3.12. *Let A be a partial assignment to variables in V_0 as defined at the beginning of this subsection. For any partial assignment to the $(3,1)$ and $(2,2)$ variables, we can assume that the assignment to the $(4,0)$ variables that minimizes the weight of violated constraints either assigns the $(4,0)$ variables according to majority, or assigns the $(4,0)$ variables according to the negation of majority.*

Proof. Fix an assignment to all variables of type $(3,1)$ or $(2,2)$. We start by analyzing the majority assignment to the $(4,0)$ variables. Observe that the $(7,0,0)$ functions under the majority assignment will still have 7 good paths through them under this assignment. The $(6,0,1)$ functions will have at least 2 good paths going through them, and the $(1,0,6)$ function will still have at least 1 good path going through them.

Every time we change the assignment of one of those $(4,0)$ variables, we will introduce 8 bad paths (actually we flip those assignments in pairs — the variable and its negation — and we will get 16 in each step.) The key is to bound the number of variables in V_1 that are now switched. Suppose we flipped $0 < k < 14$ of the $(4,0)$ functions, in the best case we have made k of the $(1,0,6)$ functions switched, and $6k/2 = 3k$ of the $(6,0,1)$ functions switched. Note that unless we flip all $(4,0)$ functions, we will never make the $(7,0,0)$ functions switch. Therefore as long as $0 < k < 14$, we will increase the number of bad paths by $8k$ but will only get at most $4k$ more switched functions. This means that unless we flip all assignments of $(4,0)$ functions to anti-majority, the value of the assignment will never be better than having all $(4,0)$ functions be assigned to majority. □

Next we show that suppose the $(4,0)$ functions are all fixed to majority, then the best assignment will assign majority to the $(3,1)$ functions.

Proposition 3.13. *Let A be a partial assignment to variables in V_0 as defined at the beginning of this subsection. If all $(4,0)$ variables in V_2 are assigned according to majority, then to complete this assignment and minimize the weight of violated constraints, one should set all $(3,1)$ variables according to majority.*

Proof. We follow the general approach of comparing the number of bad paths introduced by each flip against the potential number of switched function we get. Note that of all the types of V_1 functions we have, after setting $(4,0)$ variables to majority, the only potential function that could switch has type $(3,1,3)$ or $(0,4,3)$. However, flipping some $(3,1)$ to non-majority will only give good paths to $(0,4,3)$ and potentially decrease the number of switched functions. To get an upper-bound on the number of switched function, we can safely ignore the affect of those functions and only focus on the $(3,1,3)$ functions.

Now suppose that we change k of the $(3,1)$ functions to anti-majority assignment. Then we get $4k$ extra bad paths, and the number of switched functions will increase by at most $3k/3 = k < 4k/2$

(every flip impacts 3 $(3,1,3)$ functions, and each $(3,1,3)$ needs 3 flips such that the number of good paths becomes 0.) This means that in this case anything another than the majority assignment to the $(3,1)$ functions will give a worse assignment. □

Finally, we show that flipping all $(4,0)$ functions to anti-majority is not a good idea. The number of bad paths before assigning the $(3,1)$ and $(2,2)$ functions in V_2 is $14 \cdot 8 = 112$. With respect to the assignment where $(4,0)$ functions are assigned anti-majority and $(3,1)$ functions are assigned majority, we have 2 functions in V_1 of type $(0,7,0)$, 42 functions of type $(4,2,1)$, 14 functions of type $(0,1,6)$, 56 of type $(3,1,3)$ and 14 of type $(0,4,3)$.

We now use an argument similar as above to show that all $(3,1)$ functions should be assigned according to majority. Essentially, every flip of a $(3,1)$ function will decrease the number of good paths in 3 $(3,1,3)$ functions and 3 $(4,2,1)$ functions. After flipping k of the $(3,1)$ variables away from majority, the increase in the number of switched function is at most $3k/3 + 3k/4 = 7k/4 < 4k/2$, not enough to make up the increased number of bad paths. If we assign majority to the $(3,1)$ functions, then we already have 224 bad paths and 336 undetermined paths, thus the total number of bad paths regardless of the assignment to the $(2,2)$ functions will be $224 + 336/2 = 392$, and the number of switched function is at most 16, giving a minimum cost of $392 - 2 \cdot 16 = 360$.

On the other hand, if we take the all majority assignment, we get 280 bad paths and at most 14 switched function, giving a minimum cost of $280 - 2 \cdot 14 = 252$.

This completes the proof of the lemma. □

3.2 Assignments at distance 2 from Had_3

In this section, we prove Lemma 3.6

Proof. Now let A be a partial assignments to variables in V_0, such that there exists $x_0 \in \{-1,1\}^3$, and $l_0, l_1 \in V_0$, $l_0 \neq \pm l_1$, such that $A(f) = f(x_0)$ for all $f \in V_0 \setminus \{\pm l_0, \pm l_1\}$, and $A(f) = -f(x_0)$ otherwise. Note that there are multiple choices of x_0 and the identity of l_0 and l_1 depends on the choice of x_0. Here we pick x_0 arbitrarily from all the possible options. We call $\pm l_0$ and $\pm l_1$ corrupted affine functions. We want to show that the total weight violated by A is at least 288.

The functions in V_2 are of three types: the ones that contain neither of l_0 and l_1 as associated linear functions, the ones that contain one of them and the ones that contain both. Functions of the first and the third type are all $(3,1)$ functions, and functions of the second type can be either $(4,0)$ or $(2,2)$.

The following proposition classifies variables in V_1 and V_2.

Proposition 3.14.

- *All affine functions have 1* $(7,0,0)$ *neighbor, 3* $(1,2,4)$ *neighbors and 4* $(4,0,3)$ *neighbors. This gives a total of 16* $(7,0,0)$ *functions, 48* $(1,2,4)$ *functions and 64* $(4,0,3)$ *functions.*
- *All* $(7,0,0)$ *functions have 4* $(4,0)$ *neighbors and 3* $(3,1)$ *neighbors.*
- *All* $(1,2,4)$ *functions get 1 good path from a* $(3,1)$*, 2 bad paths from 2* $(3,1)$*'s, and 4 undetermined paths from 4* $(2,2)$*'s.*
- *All* $(4,0,3)$ *functions get 1 good path from a* $(4,0)$*, 3 good paths from 3* $(3,1)$*'s, and 3 undetermined paths from 3* $(2,2)$*'s.*

Proof. We first identify the $(4,0)$ functions.

Since these functions are associated with exactly one corrupted affine function, we start with corrupted affine functions. Let $a_0 \in V_0$ be some arbitrary corrupted affine function. After flipping two bits in a_0, the function will be at distance 2 from a_0, 6 from $-a_0$. For it to be $(4,0)$, it must have distance 4 from the other two corrupted affine functions. Let a_1 be the other corrupted function with $a_0(x_0) = a_1(x_0)$. There are 4 bits on which a_0 and a_1 differ, and let y_0 be an arbitrary one of them. Let f be the level-2 function where we flip both x_0 and y_0 in a_0. By the argument above, the only corrupted affine function associated with f is a_0. Observe also that the assignments to its associated linear functions all agree, therefore f is a $(4,0)$ function. Since each corrupted affine function a gives us 4 $(4,0)$ functions associated only with a, this gives a total of exactly 16 $(4,0)$ functions.

We now consider the paths from a_0 to f. Let $g \in V_1$ be the function obtained by flipping x_0 of a_0. If we flip one of the four y_0's discussed above, we will arrive at a $(4,0)$ function. This contributes 4 good paths through g. The other 3 paths will lead us to a level-2 function associated with 2 corrupted affine functions and 2 uncorrupted affine functions. Since the inputs assigned to the 2 uncorrupted affine functions are the same as the one assigned to a_0 but different from the one assigned to the other corrupted affine function, this gives a $(3,1)$ function with a good path through g. Summing up, g is a $(7,0,0)$ function with 4 $(4,0)$ neighbors and 3 $(3,1)$ neighbors.

Starting from a_0, consider flipping one of the 3 bits other than x_0 where a_0 and a_1 agree. Let $g \in V_1$ be a function obtained this way. We know from the above that if we further flip x_0, then we will arrive at a $(3,1)$ function associated with a_0 and $-a_1$ which contributes a good path to g. Otherwise, if we flip the remaining 2 bits where f and a_1 agree that is not x_0, then we reach a level 2 function again associated with a_0 and $-a_1$. This is a $(3,1)$ function but it contributes a bad path to g. If we instead start from g and flip one of the 4 bits where a_0 and a_1 disagree, then we arrive at a level-2 function that is only associated with a_0. Since we did not flip x_0 along the path, the function we reached is a $(2,2)$ function. We conclude that g is actually a $(1,2,4)$ function, with 1 good path from a $(3,1)$ function, 2 bad paths from 2 $(3,1)$ functions, and 4 undetermined paths from 4 $(2,2)$ functions. We also know that all corrupted affine functions have 3 such $(1,2,4)$-neighbors.

Now consider start at a_0 and flip one of the 4 bits on which a_0 and a_1 disagree, and again let g be the function we get. If we then choose to flip one of the 3 bits on which a_0 and a_1 agrees other than x_0, we will reach a $(2,2)$ function associated with only a_0, and if we flip x_0, or we will reach a $(4,0)$ function associated only with a_0. If instead we flip any of the remaining 3 bits on which a_0 and a_1 disagree, then we get a $(3,1)$ function associated with both a_0 and a_1, contributing good paths to g. This means that g is a $(4,0,3)$ function, getting 1 good path from a $(4,0)$ function and 3 good paths from 3 $(3,1)$ functions.

We have now completed the proof for corrupted affine functions. Now we turn to the uncorrupted ones. Although the idea is very similar, we need to choose the bits we flip differently.

Let a_0 now be an uncorrupted affine function. There are two corrupted affine function that disagrees with a_0 on x_0, denoted as a_1 and a_1', and these three functions are at distance 4 from each other. This means that:

1. There are two bits on which the two corrupted function agrees but they disagree with a_0, one of them is x_0, and we denote the other as y_0;

2. There are two bits on which all three functions agree, denoted as z and z'.

We first flip y_0 of a_0 and denote the resulting function by g. From g, we choose one of a_1 and a_1' and try to find path to it. Without loss of generality let us assume that we want to find a path from f to a_1. Note that f and a_1 still differs on 3 bits, one of them is x_0. We can choose to flip one

of the other two bits. This gives us a function in V_2 associated with a_1 but disagrees with a_1 on x_0, and this is actually a $(4,0)$ function. To summarize, from g, there are two choices of destination (a_1 and a_1') and two choices of the next bit to flip that will lead us to a $(4,0)$ function. Thus, function g also has 4 $(4,0)$ neighbors. For the remaining neighbors of g, consider flipping x_0. This leads to a $(3,1)$ function associated with 2 corrupted function, and which contributes a good path to g. Also, if we flip any of the two remaining bits, we would reach a $(3,1)$ function not associated with any corrupted affine functions, and this also contributes a good path to g. We conclude that g is a function of type $(7,0,0)$ with 4 $(4,0)$ neighbors and 3 $(3,1)$ neighbors.

Let us now consider what happens if we flip one of the four bits on which a_1 and a_1' disagree. Let the resulting function be g. Then we are moving closer to one of a_1 and a_1' and away from the other. Suppose without lost of generality that $d_H(g,a_1) = 3$ and $d_H(g,a_1') = 5$. There are 3 bits on which g and a_1 differ, and let z'' be such that $g(z'') = a_1'(z'')$. If we flip z'', then the resulting function will be associated with a_1 and $-a_1'$, and thus is a $(3,1)$ function contributing a good path to g. The other two bits on which g and a_1 differ are x_0 and y_0. Flipping x_0 will give us a $(2,2)$ function, and flipping y_0 will give us a $(4,0)$ function. If we instead flip either of z and z', then we get a function where the only associated corrupted affine function is $-a_1'$, and this is a $(4,0)$ function since we have not flipped x_0 yet. Flipping the remaining two bits in g will all give us $(3,1)$ functions contributing good paths to g. Thus, we conclude that g is in fact a $(4,0,3)$ function.

Let us now consider what happens if we flip x_0 of a_0. Further flipping y_0 will lead to a $(3,1)$ function that contributes a good path. Flipping either z and z' gives us $(3,1)$ functions contributing bad paths to g. Otherwise we have to flip a bit where a_1 and a_1' disagree and this would give us a $(2,2)$ function. Therefore, such g has type $(1,2,4)$.

Finally, consider flipping z or z' first. If we then flip the other one of z and z', we get a function associated with $-a_1$ and $-a_1'$, and this is a $(3,1)$ function contributing bad paths. If we flip x_0, then we get a $(3,1)$ function not associated with corrupted functions contributing bad paths. If we flip y_0, then we again get a $(3,1)$ function not associated with corrupted functions, but contributes 1 good path. If we flip one of the four bits on which a_1 and a_1' disagrees, we will get to a $(2,2)$ function. This means that such g is also $(1,2,4)$.

This completes the proof. □

Running the above argument from functions in V_2, we have the following statement for functions in V_2.

Proposition 3.15.

- *All* $(4,0)$ *functions have 4 neighbors of type* $(7,0,0)$ *and 4 of type* $(4,0,3)$.
- *All* $(3,1)$ *functions have 1 neighbors of type* $(7,0,0)$*, 4 of type* $(4,0,3)$*, 2 of type* $(1,2,4)$ *getting bad paths and 1 of type* $(1,2,4)$ *getting a good path.*

Proof. Consider first the $(4,0)$ functions. Let f be a function of type $(4,0)$. It is associated with exactly one corrupted affine function that disagrees with the other three uncorrupted affine function on x_0. Let a_0 be the corrupted affine function associated with f, and let a_1 be the other corrupted affine function such that $a_0(x_0) = a_1(x_1)$. We show that for each pair of paths going from f to one of its associated affine functions, one of them goes through a $(7,0,0)$ function in V_1 and the other goes through a $(4,0,3)$ function. The case for paths from f to the associated corrupted affine function is already argued in the proof for Proposition 3.14. Now consider paths from f to some uncorrupted affine function a_2. The path from f to a_2 consists of flipping the bit other than x_0 where a_0 and a_1 agree but not with a_2, and a bit where a_1 and a_2 agree but not with a_0. From

the proof of Proposition 3.14, we see that flipping the former bit gives us a $(4,0,3)$ function, while flipping the latter bit gives us a $(7,0,0)$ function. This completes our proof for the $(4,0)$ functions.

The $(3,1)$ functions are either associated with no corrupted affine functions, or associated with two. We consider the two cases separately.

Let f be a $(3,1)$ function that is not associated with any corrupted affine functions. Denote the associated affine functions by a_0, a_1, a_2 and a_3. Without loss of generality assume that $a_0(x_0) = a_1(x_0) = a_2(x_0) = -a_3(x_0)$. Let a and a' be the two corrupted affine functions such that $f(x_0) = -a(x_0) = -a'(x_0)$. Since f, a and a' are at distance 4 from each other, there is exactly one other input $z_0 \neq x_0$ where $f(z_0) = -a(z_0) = -a'(z_0)$. The two paths from f to a_3 are all bad. Since the only type of function in V_1 that contains bad paths is $(1,2,4)$, these two paths go through $(1,2,4)$ functions in V_1 and contributes bad paths.

Let us now flip $f(z_0)$ and get to some $g \in V_1$. Now $d_H(g,a) = d_H(g,a') = 3$, but since f is not associated with corrupted affine functions, to get to the nearest affine function from g, we need to increase the distances $d_H(g,a)$ and $d_H(g,a')$ by flipping some bit on which g, a and a' all agree, and we denote it as y_0. Let $h \in V_0$ be the function we get. We have $h(x_0) = -a(x_0) = -a'(x_0)$, $h(y_0) = -a(y_0) = -a'(y_0)$ and $h(z_0) = a(z_0) = a'(z_0)$. From the proof of Proposition 3.14, we conclude that g has type $(7,0,0)$, and if starting from f we first flip y_0 and then z_0, we will go through a $(1,2,4)$ function and contribute a good path.

Now suppose f is a $(3,1)$ function associated with two corrupted affine functions a and a'. Let the other two affine functions be a_1 and a_2. There are two subcases depending on whether $a(x_0) = a'(x_0)$.

First, suppose that $a(x_0) = a'(x_0)$. Then it must be that $a_1(x_0) \neq a_2(x_0)$. Assume that a_2 is the function with $a_2(x_0) = a(x_0) = a'(x_0)$. Then the two paths from f to a_2 are all bad paths, and thus they go through two $(1,2,4)$ functions. On the paths from f to a_1, one of the bits flipped is x_0, and the other is the other bit where a_1 disagrees with a and a'. Again, by the proof of Proposition 3.14, we conclude that one of them is a $(1,2,4)$ function and the other is a $(7,0,0)$ function.

Next, suppose that $a(x_0) = -a'(x_0)$. Then $a_1(x_0) = a_2(x_0)$. Without loss of generality, further assume that $a(x_0) = a_1(x_0) = a_2(x_0)$. Then the two paths from f to a are bad, giving f two $(1,2,4)$ neighbors. On the paths from f to a', we have to flip x_0 and another bit on which a and a' agree. We can flip the two bits in two different orders and this gives again a $(7,0,0)$ function and a $(1,2,4)$ function.

So far, we proved that no matter what kind of $(3,1)$ function we have, they all have 1 $(7,0,0)$ neighbor and 3 $(1,2,4)$ neighbor. By simple counting, we conclude that we have identified all the $(7,0,0)$ and $(1,2,4)$ functions, and therefore all remaining neighbors have type $(4,0,3)$. □

Suppose we flip k of the $(4,0)$ functions and k of their negations away from majority. At most $8k$ functions of type $(4,0,3)$ will switch, and $(7,0,0)$ functions will not switch unless $4 \leq k \leq 8$. This means that for a fixed assignment to the $(3,1)$ and $(2,2)$ functions, the weight of violated constraints will not be less than that of assigning majority to all $(4,0)$ functions, unless $k \geq 4$.

Further, observe that if two $(7,0,0)$ functions f and g share a $(4,0)$ neighbor, then $d_H(f,g) = 2$. Therefore they share exactly $2! = 2$ $(4,0)$ neighbors. Thus if $k = 4,5$, at most 1 $(7,0,0)$ function and its negation will be switched.

Let us consider the case when $k = 4$. We are flipping 8 $(4,0)$ functions, generating 64 bad paths. This should be compensated by turning at least 32 functions from the set of $(7,0,0)$ and $(4,0,3)$ functions into functions that have no good paths. Suppose we choose $0 \leq k_1 \leq 1$ of the $(7,0,0)$ functions and $0 \leq k_2 \leq 64$ of the $(4,0,3)$ functions, where $k_1 + k_2 = 32$. Each $(7,0,0)$ is connected to a set of 3 $(3,1)$ functions, and these sets are disjoint for different $(7,0,0)$ functions. Therefore, at least $3k_1$ of the $(3,1)$ functions need to be flipped. Also, each $(4,0,3)$ is connected

to a set of 3 $(3,1)$ functions, and each $(3,1)$ function is connected to 4 $(4,0,3)$ functions, therfore at least $3k_2/4$ $(3,1)$ functions need to be flipped. Moreover, the number of functions flipped must be even because we are flipping both the function and its negation. Combining this, we conclude that at least 24 of the $(3,1)$ needs to be flipped, contributing 96 new bad paths. This means that we now have $64+96=160$ bad paths and would like to find at least 80 functions in V_1 that will potentially switch. This is the total number of $(7,0,0)$ and $(4,0,3)$ functions, and since we assumed that we only switch 1 of the $(7,0,0)$ functions, which means that it is already impossible.

For $k>4$, the number of functions we need to potentially switch will only be higher, and therefore flipping $(4,0)$ is never optimal.

Once we have that the $(4,0)$ functions should be assigned majority, the rest of the argument is easy. The only function that could possibly switch are the $(1,2,4)$ functions. However, when we flip a $(3,1)$, we get 4 bad paths and could expect to gain at most 1 switched function. Therefore flpping $(3,1)$ in this case will always increase the weight of violated constraints.

It is then easy to see that under the majority assignment, there is no switched function, and the total number of bad paths is 288. □

4 Optimality of the 11/8-gadget

In this section, we will construct an optimal solution to the dual LP given in Definition 2.28. This yields the following theorem.

Theorem 4.1. *[Theorem 1.8 restated] The value of the LP in Definition 2.28 is $\frac{11}{64}$. As a result, for every (c,s)-gadget reducing Max-Had_3 to Max-2-Lin(2),*

$$\frac{s}{c} \leq \frac{11}{8}.$$

In other words, the gadget given in Theorem 3.1 is optimal *among gadget reductions from Chan's 7-ary Hadamard predicate.*

Proof. Our goal is to construct $\mathcal{A} \in \mathsf{R}(\mathsf{Had}_k)$, i.e. a folded distribution of assignments which is random on the primary variables. For $i \in \{0,1,2\}$, denote by $\mathsf{R}_i(\mathsf{Had}_3)$, the set of distributions $\mathcal{A}_i$ such that the string

$$\left(A\left(\chi_{\emptyset}\right), A\left(\chi_{\{1\}}\right), A\left(\chi_{\{2\}}\right), A\left(\chi_{\{3\}}\right), A\left(\chi_{\{1,2\}}\right), A\left(\chi_{\{1,3\}}\right), A\left(\chi_{\{2,3\}}\right), A\left(\chi_{\{1,2,3\}}\right)\right)$$

is, over a random $A \sim \mathcal{A}_i$, distributed like a uniformly random element of $\{-1,1\}^8$ which is distance i from satisfying the Had_3 predicate. To prove Theorem 4.1, we will construct three separate distributions $\mathcal{A}_0$, $\mathcal{A}_1$, and $\mathcal{A}_2$ with the property that $\mathcal{A}_i \in \mathsf{R}_i(\mathsf{Had}_3)$ for each $i \in \{0,1,2\}$. Then, using Proposition 2.33, we will set $\mathcal{A} = \frac{1}{16}\mathcal{A}_0 + \frac{1}{2}\mathcal{A}_1 + \frac{7}{16}\mathcal{A}_2$.

Using Proposition 2.29, we can decompose $\{-1,1\}^8 = V_0 \cup V_1 \cup V_2$. By Proposition 2.31, a length-one edge (x,y) in $\{-1,1\}^8$ has one of two types: either it goes between V_0 and V_1, or it goes between V_1 and V_2. Each $\mathcal{A}_i$ distribution we construct will perform equally well on edges of a given type. As a result, for each distribution $\mathcal{A}_i$, we only need to keep track of two numbers, one for each edge type. To do so, for any fixed edge (x,y) of type $V_0 \leftrightarrow V_1$, define

$$\mathsf{val}(\mathcal{A}', V_0 \leftrightarrow V_1) := \mathop{\mathbf{Pr}}_{A \sim \mathcal{A}'}[A(x) = A(y)],$$

and define $\mathsf{val}(\mathcal{A}', V_1 \leftrightarrow V_2)$ analogously. For convenience in this section, we will keep track of these $\mathsf{val}(\cdot)$ parameters rather than the corresponding $\mathsf{uval}(\cdot)$ parameters.

For each $i \in \{0,1,2\}$, our goal is to construct the $\mathcal{A}_i$ which strikes the right balance between making $\mathsf{val}(\mathcal{A}_i, V_0 \leftrightarrow V_1)$ large and making $\mathsf{val}(\mathcal{A}_i, V_1 \leftrightarrow V_2)$ large. The next three lemmas show what we achieve.

Lemma 4.2. *There exists a distribution $\mathcal{A}_0 \in \mathsf{R}_0(\mathsf{Had}_3)$ such that*

$$\mathsf{val}(\mathcal{A}_0, V_0 \leftrightarrow V_1) = \frac{7}{8} \quad \textit{and} \quad \mathsf{val}(\mathcal{A}_0, V_1 \leftrightarrow V_2) = \frac{7}{8}.$$

Lemma 4.3. *There exists a distribution $\mathcal{A}_1 \in \mathsf{R}_1(\mathsf{Had}_3)$ such that*

$$\mathsf{val}(\mathcal{A}_1, V_0 \leftrightarrow V_1) = \frac{25}{32} \quad \textit{and} \quad \mathsf{val}(\mathcal{A}_1, V_1 \leftrightarrow V_2) = \frac{7}{8}.$$

Lemma 4.4. *There exists a distribution $\mathcal{A}_2 \in \mathsf{R}_2(\mathsf{Had}_3)$ such that*

$$\mathsf{val}(\mathcal{A}_2, V_0 \leftrightarrow V_1) = \frac{7}{8} \quad \textit{and} \quad \mathsf{val}(\mathcal{A}_2, V_1 \leftrightarrow V_2) = \frac{43}{56}.$$

By setting $\mathcal{A} := \frac{1}{16}\mathcal{A}_0 + \frac{1}{2}\mathcal{A}_1 + \frac{7}{16}\mathcal{A}_2$, we see that

$$\mathsf{val}(\mathcal{A}, V_0 \leftrightarrow V_1) = \mathsf{val}(\mathcal{A}, V_1 \leftrightarrow V_2) = \frac{53}{64}.$$

Finally, $1 - \frac{53}{64} = \frac{11}{64}$, and this is the number guaranteed by the theorem.

We now prove Lemmas 4.2, 4.3, and 4.4. The first two are straightforward, but Lemma 4.4 is relatively involved.

Proof of Lemma 4.2. The distribution $\mathcal{A}_0$ simply picks a random (negated) dictator assignment. More formally, it does the following:

1. Pick $b \in \{-1, 1\}$ and $i \in \{1, 2, \ldots, K\}$ independently and both uniformly at random.
2. Let d_i be the i-th dictator function.
3. Output the function $b \cdot d_i$.

A random dictator will satisfy any fixed edge of the hypercube with probability $\frac{7}{8}$. Thus, it remains to prove that $\mathcal{A}_0 \in \mathsf{R}_0(\mathsf{Had}_3)$, and this is easy to check. □

Proof of Lemma 4.3. The distribution $\mathcal{A}_1$ picks a random (negated) dictator assignment and corrupts its value on a single primary variable. More formally, it does the following:

1. Sample A from $\mathcal{A}_0$.
2. Let x be a uniformly random primary variable.
3. Set $\widetilde{A}(x) := -A(x)$ and $\widetilde{A}(-x) := -A(-x)$. Set $\widetilde{A} := A$ for all other inputs.
4. Output $\widetilde{A}$.

Note that whenever $\widetilde{A}$ is generated from A, $\widetilde{A}$ and A agree on all variables in V_1 and V_2. As a result,

$$\mathsf{val}(\mathcal{A}_1, V_1 \leftrightarrow V_2) = \mathsf{val}(\mathcal{A}_0, V_1 \leftrightarrow V_2) = \frac{7}{8}$$

Next, fix an edge (x, y) where $x \in V_0$ and $y \in V_1$. Then $\widetilde{A}$ and A agree on (x, y) unless either x or $-x$ is selected as the primary variable in Step 2, in which case they disagree on x. By Proposition 2.29, there are 16 primary variables, meaning this event occurs with probability $\frac{1}{8}$. As a result,

$$\mathsf{val}(\mathcal{A}_1, V_0 \leftrightarrow V_1) = \frac{7}{8} \cdot \mathsf{val}(\mathcal{A}_0, V_0 \leftrightarrow V_1) + \frac{1}{8} \cdot (1 - \mathsf{val}(\mathcal{A}_0, V_0 \leftrightarrow V_1)) = \frac{25}{32}.$$

It remains to check that $\mathcal{A}_1 \in \mathsf{R}_1(\mathsf{Had}_3)$; this follows from the fact that $\mathcal{A}_0 \in \mathsf{R}_0(\mathsf{Had}_3)$ and that $\widetilde{A}$ is chosen by randomly perturbing A on a single primary variable. □

Proof of Lemma 4.4. The distribution $\mathcal{A}_2$ is itself a mixture of two other distributions, $\mathcal{A}_2^0$ and $\mathcal{A}_2^1$. We now describe them separately and combine them later.

Constructing $\mathcal{A}_2^0$: The distribution $\mathcal{A}_2^0$ is generated by the following process:

1. Sample d_i uniformly at random from the set of dictators and negated dictators.
2. Form $\widetilde{d_i}$ by negating d_i on two uniformly random primary variables (and their negations).
3. Set $A : V \to \{-1, 1\}$ to agree with $\widetilde{d_i}$ on V_0.
4. For every $y \in V_1$, let x be y's neighbor in V_0.
   - If $A(x) = d_i(x)$, set $A(y) := d_i(y)$.
   - Otherwise, set $A(y) := A(x)$.
5. For every $z \in V_2$, set $A(z)$ to the majority value of A on z's eight neighbors in V_1. (Ties will not occur.)

By the construction of A, it is immediate that $\mathcal{A}_2^0 \in \mathsf{R}_2(\{-1, 1\}^8)$. As for its performance on the two edge types, we have the following proposition.

Proposition 4.5. $\mathsf{val}(\mathcal{A}_2^0, V_0 \leftrightarrow V_1) = \frac{29}{32}$ *and* $\mathsf{val}(\mathcal{A}_2^0, V_1 \leftrightarrow V_2) = \frac{167}{224}$.

Proof. Let $x \in V_0$ and $y \in V_1$ be neighbors. With probability $\frac{3}{4}$, $A(x) = d_i(x)$, in which case $A(y) = d_i(y)$. Conditioning on this event, $A(y) = A(x)$ with $\frac{7}{8}$ probability,. as the dictator assignment satisfies each edge with probability $\frac{7}{8}$. On the other hand, when $A(x) \neq d_i(x)$, which happens with probability $\frac{1}{4}$, $A(y)$ always equals $A(x)$. Thus,

$$\mathsf{val}(\mathcal{A}_2^0, V_0 \leftrightarrow V_1) = \frac{3}{4} \cdot \frac{7}{8} + \frac{1}{4} = \frac{29}{32}.$$

To compute $\mathsf{val}(\mathcal{A}_2^0, V_1 \leftrightarrow V_2)$, we will condition the output of $\mathcal{A}_2^0$ on the choice of the (possibly negated) dictator d_i in Step 1. Let us focus on a particular $z \in V_2$. We will now describe what the immediate neighborhood of z looks like.

Let $\mathcal{N}(z) \subseteq V_1$ denote the neighborhood of z, and let x_1, x_2, x_3, and x_4 be the four points in V_0 which are distance two from z. Regardless of what d_i is, $(d_i(x_1), d_i(x_2), d_i(x_3), d_i(x_4))$ will always have a majority size of three. Furthermore, $d_i(z)$ agrees with this majority. Let $\widetilde{x} \in \{x_1, x_2, x_3, x_4\}$ be the variable in the minority, and assume WLOG that $\widetilde{x} = x_1$. Then of the two elements in $\mathcal{N}(z)$ which neighbor x_1, d_i assigns one a value of 1 and the other a value of (-1); for the remaining elements $y \in \mathcal{N}(z)$, $d_i(y) = d_i(z)$. A pictorial representation of this is given in Figure 2.

Consider the two primary variables which were selected in Step 2. If a given primary variable x_j is selected, then the value given to it by d_i is negated when forming A. Furthermore, all of x_j's

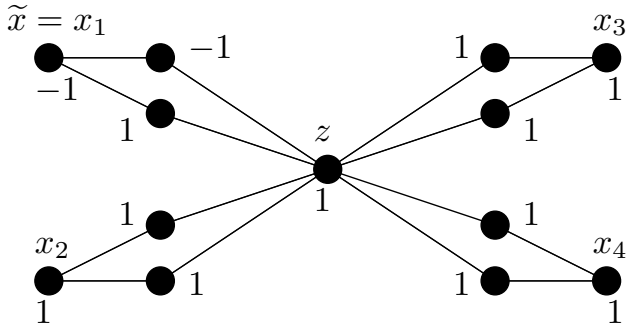

Figure 2: The neighborhood of z. The variables are labeled with the assignment d_i gives them. Note that the assignment of z agrees with all but one of its neighbors and all but one of the x_j's. Further, the two variables which disagree with z are adjacent.

neighbors in V_1 are assigned the value $A(x_j)$. Pictorially, if x_3 in Figure 2 was selected in Step 2, then its and its two neighbors' values would be flipped to (-1) in A. Similarly, if $\widetilde{x} = x_1$ in Figure 2 was selected, then both it and its two neighbors would be given the value 1 by A.

The value A assigns to z is just the majority value of z's neighbors, so A's success on the edges neighboring z will just be the fractional size of this majority. If x_1 is selected in Step 2, then the value given to every $y \in \mathcal{N}(z)$ agrees with the value given to y's neighbor in the x_j's. Thus, in this case, the majority size of z's neighbors will just be $2 \cdot \textsf{Maj-Size}(A(x_1), A(x_2), A(x_3), A(x_4))$. Otherwise, the two neighbors of x_1 will disagree with one another, and the majority size will be $1 + 2 \cdot \textsf{Maj-Size}(A(x_2), A(x_3), A(x_4))$.

Now, we split into cases based on how many elements of $\{x_1, x_2, x_3, x_4\}$ were selected in Step 2. Case 0 occurs when zero were selected, Case 1 when one was selected, and so forth.

Case 0: In this case x_1 is not selected, and Maj-Size of the remaining x_j's is three. As a result, the success probability is $\frac{1+2\cdot 3}{8} = \frac{7}{8}$.

Case 1: We split into two subcases depending on whether x_1 was selected.

Case x_1 is selected: In this case, $\textsf{Maj-Size}(A(x_1), A(x_2), A(x_3), A(x_4)) = 4$, and so the success probability is 1.

Case x_1 is not selected: In this case, $\textsf{Maj-Size}(A(x_2), A(x_3), A(x_4)) = 2$, and so the success probability is $\frac{1+2\cdot 2}{8} = \frac{5}{8}$.

The first subcase occurs with probability $\frac{1}{4}$ and the second subcase occurs with probability $\frac{3}{4}$. Combined, this means that A succeeds with probability $\frac{1}{4} + \frac{3}{4} \cdot \frac{5}{8} = \frac{23}{32}$ in this case.

Case 2: Again, we split into two subcases depending on whether x_1 was selected.

Case x_1 is selected: In this case, $\textsf{Maj-Size}(A(x_1), A(x_2), A(x_3), A(x_4)) = 3$, so the success probability is $\frac{2\cdot 3}{8} = \frac{6}{8}$.

Case x_1 is not selected: In this case, $\textsf{Maj-Size}\,(A(x_2), A(x_3), A(x_4)) = 2$, and so the success probability is $\frac{1+2\cdot 2}{8} = \frac{5}{8}$.

Both subcases occur with probability $\frac{1}{2}$. As a result, A succeeds in this case with probability $\frac{1}{2}\left(\frac{3}{4} + \frac{5}{8}\right) = \frac{11}{16}$

There are $\binom{8}{2}$ possible choices for the two primary variables which were negated to form $\widetilde{d}_i$. Of these, $\binom{4}{2}$ yield Case 0, $4 \cdot 4$ yield Case 1, and $\binom{4}{2}$ yield Case 2. This gives a total success probability of

$$\mathsf{val}(\mathcal{A}_2^0, V_1 \leftrightarrow V_2) = \frac{\binom{4}{2}}{\binom{8}{2}} \cdot \frac{7}{8} + \frac{4 \cdot 4}{\binom{8}{2}} \cdot \frac{23}{32} + \frac{\binom{4}{2}}{\binom{8}{2}} \cdot \frac{11}{16} = \frac{167}{224},$$

as guaranteed by the proposition. □

Constructing $\mathcal{A}_2^1$: In constructing $\mathcal{A}_2^1$, it will be convenient to take the viewpoint of $V = \{f \mid f : \{-1,1\}^3 \to \{-1,1\}\}$. The distribution $\mathcal{A}_2^1$ is generated by the following process:

1. Pick $x_1, x_2, x_3 \in \{-1,1\}^3$.
2. Set $(x_4)_i = (x_1)_i \cdot (x_2)_i \cdot (x_3)_i$ for $i \in [3]$.
3. Pick $b_1, \ldots, b_4 \in \{-1,1\}$ subject to $b_1 b_2 b_3 b_4 = -1$.
4. Set $A_1, \ldots, A_4$ such that $A_i(f) = b_i \cdot f(x_i)$ for $i \in [4]$.
5. Pick a uniformly random $M : \{-1,1\}^4 \to \{-1,1\}$ subject to
   - $M(a_1, \ldots, a_4) = \mathrm{sgn}(a_1 + \ldots + a_4)$ when $a_1 + \ldots + a_4 \neq 0$, and
   - $M(a_1, \ldots, a_4) = -M(-a_1, \ldots, -a_4)$ otherwise.
6. Output $A(f) = M(A_1(f), \ldots, A_4(f))$.

The construction of $\mathcal{A}_2^1$ is similar to the construction of the variables in V_2 in the proof of Proposition 2.32. Via the correspondence shown in the proof of Proposition 2.33, Proposition 2.32 shows that $\mathcal{A}_2^1 \in \mathsf{R}_2(\mathsf{Had}_3)$. As for its performance on the two edge types, we have the following proposition.

Proposition 4.6. $\mathsf{val}(\mathcal{A}_2^1, V_0 \leftrightarrow V_1) = \mathsf{val}(\mathcal{A}_2^1, V_1 \leftrightarrow V_2) = \frac{13}{16}$.

Proof. Let $f, f' : \{-1,1\}^3 \to \{-1,1\} \in V$ be neighboring variables, i.e. they differ on one input. Let $\tilde{x}$ be the input that f and f' differ on. Let us condition on whether $\tilde{x} \in \{x_1, \ldots, x_4\}$. As there are eight elements of $\{-1,1\}^3$, both cases occur with half probability.

Case $\tilde{x} \notin \{x_1, \ldots, x_4\}$: In this case, $A_i(f) = A_i(f')$ for all i, so $A(f)$ always equals $A(f')$.

Case $\tilde{x} \in \{x_1, \ldots, x_4\}$: In this case, one of the two strings $(f(x_1), \ldots, f(x_4))$ and $(f'(x_1), \ldots, f'(x_4))$ has an even parity and the other string has an odd parity. Without loss of generality, assume that $(f(x_1), \ldots, f(x_4))$ has the odd parity.

Over the random choice of the b_i's, the string $(b_1 f(x_1), \ldots, b_4 f(x_4))$ is distributed like a uniformly random string with an even parity. In particular, the probability that all four entries are the same is $1/4$. When this occurs, then $A(f) = A(f')$. This holds because if, for instance, $(b_1 f(x_1), \ldots, b_4 f(x_4)) = (1,1,1,1)$, then $(b_1 f'(x_1), \ldots, b_4 f'(x_4))$ has three 1's, and so the majority value is 1.

This means that with $3/4$ probability, $(b_1 f(x_1), \ldots, b_4 f(x_4))$ has two 1's and two (-1)'s. Fix the b_i's, and consider the randomness over the choice of M. In this case, $A(f)$ will be a uniformly random ± 1 bit, because M is uniformly random when there is no clear majority. On the other hand, $(b_1 f'(x_1), \ldots, b_4 f'(x_4))$ will have a clear majority, so $A(f')$ a deterministic value. Thus, in this case, $A(f) = A(f')$ with half probability.

Putting it all together. As a result,

$$\mathbf{Pr}[A(f) = A(f')] = \frac{1}{2} + \frac{1}{2} \cdot \left(\frac{1}{4} + \frac{3}{4} \cdot \frac{1}{2} \right),$$

which equals 13/16. As this holds for any neighboring functions f and f', we get our desired conclusion, which is that $\mathsf{val}(\mathcal{A}_2^1, V_0 \leftrightarrow V_1) = \mathsf{val}(\mathcal{A}_2^1, V_1 \leftrightarrow V_2) = 13/16$. □

Combining $\mathcal{A}_2^0$ and $\mathcal{A}_2^1$: The final distribution $\mathcal{A}_2$ is given by the mixture $\mathcal{A}_2 = \frac{2}{3}\mathcal{A}_2^0 + \frac{1}{3}\mathcal{A}_2^1$. Because $\mathcal{A}_2^0$ and $\mathcal{A}_2^1$ are both elements of $\mathsf{R}_2(\mathsf{Had}_3)$, it follows that $\mathcal{A}_2 \in \mathsf{R}_2(\mathsf{Had}_3)$. Furthermore, by combining Propositions 4.5 and 4.6, we see that

$$\mathsf{val}(\mathcal{A}_2, V_0 \leftrightarrow V_1) = \frac{7}{8} \quad \text{and} \quad \mathsf{val}(\mathcal{A}_2, V_1 \leftrightarrow V_2) = \frac{43}{56},$$

as promised by the lemma. □

This completes the construction of the distributions $\mathcal{A}_0$, $\mathcal{A}_1$, and $\mathcal{A}_2$, thereby completing the theorem. □

5 A candidate factor-3/2 hardness reduction

5.1 The Game Show Conjecture

Herein we present an interesting problem concerning analysis of Boolean functions. We make a conjecture about its solution which, if true, implies NP-hardness (with quasilinear blowup) of factor-$(\frac{3}{2} - \delta)$ approximating 2-Lin(2) for any $\delta > 0$.

Definition 5.1. Let $g : \{-1,1\}^n \to \{-1,1\}$ be a folded function (i.e., $g(-x) = -x$). The *Game Show*, played with *Middle Function* g, works as follows. There are two personages: the *Host* and the *Contestant*. Before the game begins, the Host secretly picks a uniformly random path $\boldsymbol{\pi}$ from $(1,1,\dots,1)$ to $(-1,-1,\dots,-1)$ in the Hamming cube. (Equivalently, $\boldsymbol{\pi}$ is a uniformly random permutation on $[n]$.) The Host also secretly picks $\boldsymbol{T} \sim \text{Binomial}(n, \frac{1}{2})$. We define the *secret half-path* to be the sequence of the first $\boldsymbol{T}$ edges along $\boldsymbol{\pi}$: $(x_0, x_1), (x_1, x_2), \dots, (x_{\boldsymbol{T}-1}, x_{\boldsymbol{T}})$. Note that $x_{\boldsymbol{T}}$ is uniformly distributed on $\{-1,1\}^n$.

The Game now begins, with the current *time* being $t = 0$, the current *point* being $x_0 = (1,1,\dots,1)$, and the current *function* being $\widetilde{g} = g$. (The current function will always be $\pm g$.)

At each time step $t = 0, 1, 2, \dots$, the Host asks whether the Contestant would like to *negate* the current function, meaning replace $\widetilde{g}$ with $-\widetilde{g}$. If the Contestant does not negate the current function there is no cost. However, if the Contestant elects to negate the current function, the Contestant must pay a cost of

$$w(t) \coloneqq \frac{1}{(1 - t/n)^2}. \tag{3}$$

After the Contestant makes the decision, the Host reveals to the Contestant what the $(t+1)$th point on the secret half-path is, and the new time becomes $t + 1$.

As soon as time $\boldsymbol{T}$ is reached, the Game ends. At this instant, if $\widetilde{g}(x_{\boldsymbol{T}}) \neq 1$, then the Contestant incurs a further cost of $w(\boldsymbol{T})$. (It's as though the Contestant is now obliged to negate $\widetilde{g}$.) Thus one can think of the Contestant's goal throughout the Game as trying to ensure that $\widetilde{g}(x_{\boldsymbol{T}})$ will equal 1, while trying to minimize the total cost incurred by all negations.

We define $\text{cost}(g)$ to be the least expected cost that a Contestant can achieve when the Game Show is played with Middle Function g.

To get a feel for the Game Show, let's make some observations. First, as mentioned, the negation cost $w(t)$ is an increasing function of time; i.e., the later it is in the Game, the more costly it is for the Contestant to negate $\widetilde{g}$. The cost to negate at the beginning of the Game is $w(0) = 1$ and the cost to negate when the Game ends is $w(\boldsymbol{T}) \sim 4$ (with very high probability, since $\boldsymbol{T} = (\frac{1}{2} \pm o_n(1))n$ with very high probability). As a consequence, we always have $\mathrm{cost}(g) \leq 2 + o_n(1)$. The reason is that the Contestant can always use the strategy of never negating $\widetilde{g}$ unless obliged to at the end of the Game. In this case, the final evaluation will be $g(\boldsymbol{x})$ where $\boldsymbol{x}$ is uniformly random. By oddness of g, this evaluation is -1 with probability exactly $\frac{1}{2}$, and only in this case does the Contestant suffer a cost, namely $4 \pm o_n(1)$ (with high probability).

It can be shown that the best Middle Function g for the Contestant to play with is any (positive) dictator, $g = d_i$, say. It's easy to check that the Contestant's best strategy is the obvious one: negate if and only if the Host restricted coordinate i to -1 on the previous turn. To estimate the expected cost of this strategy, first note that the probability the Host restricts coordinate i throughout the course of the game is $\frac{1}{2}$. If the Host never restricts coordinate i then the Contestant will have cost 0. Otherwise, conditioned on the Host restricting coordinate i over the course of the game, the Contestant will have cost $w(\boldsymbol{t}^*)$, where $\boldsymbol{t}^*$ is "essentially" uniformly distributed on $\{1, 2, \ldots, \frac{n}{2}\}$. At this point it's natural to introduce the "continuous" time parameter $u = t/n$, which ranges in $[0, \frac{1}{2}]$ (with very high probability), as well as the function

$$W(u) \coloneqq w(un) = \frac{1}{(1-u)^2}, \tag{4}$$

which increases from 1 to 4 on $[0, \frac{1}{2}]$. The distribution of $\boldsymbol{u}^* = \boldsymbol{t}^*/n$ is "essentially" uniform on $[0, \frac{1}{2}]$. More precisely, one may check that up to $o_n(1)$ errors, the expected cost to the Contestant conditioned on coordinate i being restricted is just the average value of W, namely

$$\int_0^{\frac{1}{2}} W(u) \cdot 2du = 2.$$

Thus we finally conclude that $\mathrm{cost}(d_i) \sim \frac{1}{2} \cdot 2 = 1$.

Let's now look at the best strategy when the Middle Function is a *negated* dictator; i.e., let's try to determine $\mathrm{cost}(-d_i)$. Playing the Game Show with Middle Function $-d_i$ is more stressful for the Contestant because at the beginning of the game we have $\widetilde{g}(x) = -1$. Assume the Contestant elects not to negate $\widetilde{g}$ for a while at the beginning of the Game. If ever the Host restricts the ith coordinate to -1 then the Contestant can relax, knowing that no costs at all will be incurred. However as time progresses without the ith coordinate being restricted, the Contestant will naturally get more and more nervous that the game will end with $\widetilde{g}(1, 1, \ldots, 1) = -1$, forcing a cost of essentially 4. On the other hand, if the Contestant decides to "preemptively" negate, there is still some chance that the ith coordinate will subsequently be restricted before the game ends, forcing the Contestant to negate *again*. Actually, it's not hard to show that the best strategy for the Contestant is of the following form, for some value $u_0 \in [0, \frac{1}{2}]$: "If ever the Host restricts the ith coordinate to -1, negate $\widetilde{g}$ if necessary and then never negate again. Otherwise, wait until the continuous time hits u_0 and then negate $\widetilde{g}$." For example, the $u_0 = 0$ case of this strategy involves negating immediately at the game's beginning (incurring cost 1) and then playing the optimal strategy for a positive dictator (incurring expected cost $1 \pm o_n(1)$). The total expected cost is $2 \pm o_n(1)$. As another example, the $u_0 = \frac{1}{2}$ case of this strategy is the generic strategy of never negating unless forced to at the game's end. This case *also* has an expected cost of $2 \pm o_n(1)$. In fact, one can do a short calculation to show that the expected cost is $2 \pm o_n(1)$ for *every* value of u_0. This is by design: the particular cost function W is *the unique function with this property.*

We've concluded that $\text{cost}(d_i) \sim 1$ and $\text{cost}(-d_i) \sim 2$. To now state our conjecture about the Game Show, we need a piece of notation; for $g : \{-1,1\}^n \to \{-1,1\}$ and "negation pattern" $b \in \{-1,1\}^n$, we write g^{+b} to denote the function defined by $g^{+b}(x) = g(b_1 x_1, \ldots, b_n x_n)$. Roughly speaking, our conjecture about the Game Show is that for every odd g, the average value of $\text{cost}(g^{+b})$ over all b is at least $\frac{3}{2}$. To be precise, we need to be concerned with averaging over merely pairwise-independent distributions on b.

Game Show Conjecture. *Let $g : \{-1,1\}^n \to \{-1,1\}$ be odd and let $\mathcal{D}$ be any distribution on $\{-1,1\}^n$ which is pairwise-independent and symmetric (meaning $\mathbf{Pr}_{\mathcal{D}}[\boldsymbol{b}] = \mathbf{Pr}_{\mathcal{D}}[-\boldsymbol{b}]$). Then*

$$\mathop{\mathbf{E}}_{\boldsymbol{b} \sim \mathcal{D}}[\text{cost}(g^{+\boldsymbol{b}})] \geq \frac{3}{2} - o_n(1).$$

Our motivation for making the Game Show Conjecture is the following result:

Theorem 5.2. *Suppose the Game Show Conjecture is true. Then it is* NP*-hard to approximate* 2-Lin(2) *(and hence also* Max-Cut*) to factor $\frac{3}{2} - \delta$ for any $\delta > 0$.*

We remark that given a Middle Function g, in some sense it is "easy" to determine the Contestant's best strategy. It can done with a dynamic program, since the Game Show is essentially a 2-Lin(2) instance on a tree graph. Nevertheless, we have been unable to prove the Game Show Conjecture. We will discuss some of our efforts in Section 5.3. First, however, we will prove Theorem 5.2.

5.2 Proof of Theorem 5.2

The proof is by construction of a gadget as in Definition 2.24.

Theorem 5.3. *Suppose the Game Show Conjecture is true. Then for each k, there exists a (c,s)-gadget reducing Max-*Had_k *to Max-*2-Lin(2) *satisfying*

$$\frac{s}{c} \geq \frac{3}{2} - o_k(1).$$

Via Corollary 2.26, this shows that for every $\epsilon > 0$, it is NP-hard to approximate 2-Lin(2) to factor $\frac{3}{2} - o_k(1) - \epsilon$. Thus, by taking k large enough and ϵ small enough so that $o_k(1) + \epsilon \leq \delta$, we get Theorem 5.2.

Now we prove Theorem 5.3.

Proof of Theorem 5.3. Set $n := 2^k$ (previously we have used K for this number, but for this proof we will use n). For the gadget $\mathcal{G}$, let $\{-1,1\}^n$ denote the usual vertex set, and let Z be the set of $2n$ primary variables. For reasons that will be clearer later, we will call these variables the Contestant Strategy variables. As in the definition of a (c,s)-gadget, we need only consider assignments $A : \{-1,1\}^n \to \{-1,1\}$ which are folded.

We will add one twist to this construction: we will have an additional collection of variables, identified with $\{-1,1\}^n$, called the Middle Function variables. We will write $g : \{-1,1\}^n \to \{-1,1\}$ for assignments to these variables, and we will use folding here as well so that each g can assumed to be folded. We remark that it would actually be okay if the Middle Function variables and the Contestant Strategy variables were identified; however, we find it conceptually clearer to separate them, and it doesn't affect our analysis of gadget's quality.

We now describe the constraints we put on our gadget; these are highly reminiscent of the Game Show described in the previous section. All of the constraints are equality tests along Hamming edges (either within the Contestant Strategy/Middle Function hypercubes, or between them). We will henceforth refer to gadget variables as "points", and assume that n is at least a sufficiently large universal constant.

Overall Gadget Test $\mathcal{G}$:
With probability δ, run the Average Sensitivity Test; with probability $1-\delta$, run the Game Show Test.

Average Sensitivity Test:
Choose a uniformly random edge $(\boldsymbol{y}, \boldsymbol{y}')$ in the Middle Function hypercube and test that $g(\boldsymbol{y}) = g(\boldsymbol{y}')$.

Game Show Test:
Choose a random primary point $\boldsymbol{x}_0 \in Z$ in the Contestant Strategy cube. Choose a random path $(\boldsymbol{x}_0, \boldsymbol{x}_1), (\boldsymbol{x}_1, \boldsymbol{x}_2), \ldots, (\boldsymbol{x}_{n-1}, \boldsymbol{x}_n)$ from $\boldsymbol{x}_0$ to $\boldsymbol{x}_n := -\boldsymbol{x}_0$. Choose $\boldsymbol{T}$ to be a Binomial$(n, \frac{1}{2})$ random variable, conditioned on being in the range $\frac{n}{2} \pm \sqrt{n \log n}$. Choose $\boldsymbol{t} \in [\boldsymbol{T}]$ such that $\mathbf{Pr}[\boldsymbol{t} = t]$ is proportional to $w(t)$ (as defined in (3)). Finally, if $\boldsymbol{t} < \boldsymbol{T}$ then test that $A(\boldsymbol{x}_{\boldsymbol{t}-1}) = A(\boldsymbol{x}_{\boldsymbol{t}})$; otherwise, test that $A(\boldsymbol{x}_{\boldsymbol{t}-1}) = g(\boldsymbol{x}_{\boldsymbol{T}})$. (In the former case, both points are taken from the Contestant Strategy cube; in the latter case, $\boldsymbol{x}_{\boldsymbol{t}-1}$ is taken from the Contestant Strategy cube and $\boldsymbol{x}_{\boldsymbol{T}}$ is taken from the Middle Function cube.)

Here δ will be some decreasing function of n which we specify implicitly later. We first prove a little lemma:

Lemma 5.4. *Let $1 \leq t \leq \frac{n}{2} + \sqrt{n \log n}$ and write $u = t/n$. Then in the Game Show Test,*

$$\mathbf{Pr}[\boldsymbol{t} = t] = \frac{W(u)}{n} \cdot (1 \pm o_n(1)),$$

where W is defined as in (4).

Proof. By definition, all we need to do is show that the "constant of proportionality" in the Game Show Test, namely $\sum_{i=1}^{\boldsymbol{T}} w(t)$, is equal to $n(1 \pm o_n(1))$, uniformly for each outcome of $\boldsymbol{T}$. Since w is an increasing function, this discrete sum is bounded between the integrals $\int_0^{\boldsymbol{T}} w$ and $\int_1^{\boldsymbol{T}+1} w$. Using the fact that $\int_0^{1/2} W = 1$, it's easy to compute that both of the bounding integrals are $n \pm O(\sqrt{n \log n})$ when $\boldsymbol{T}$ is $\frac{n}{2} \pm \sqrt{n \log n}$. □

It's easy to see that in both the Average Sensitivity Test and the Game Show Test, we have complete symmetry with respect to the *direction* in $[n]$ of the edge being tested for equality. Thus, for any dictator function d_i, the probability of rejecting when $A = g = d_i$ is indeed precisely $\frac{1}{n}$. Thus, we need only show that $s \geq \frac{3/2 - o_n(1)}{n}$ for $\mathcal{G}$ to prove Theorem 5.3.

Theorem 5.5 (Soundness)**.** *Assume the Game Show Conjecture. Suppose A's assignments to the $2n$ primary points Z are chosen uniformly at random. Then regardless of how g and the remaining values of A are chosen, the probability that the Overall Gadget Test rejects is, in expectation, at least $\frac{3/2 - o_n(1)}{n}$, provided that $\delta = o_n(1)$ is suitably chosen.*

Proof. Let's ignore the Average Sensitivity Test for a moment and focus on the Game Show Test. It is evidently somewhat similar to the Game Show as played with Middle Function $g : \{-1, 1\}^n \to \{-1, 1\}$. In brief, the key differences are: (i) the Game Show Test starts its path from a random point in Z, rather than from $(1, 1, \ldots, 1)$; (ii) in the Game Show Test, the Contestant/A-chooser's job is even harder than in the Game Show, because the entire strategy A must be fixed before the Game begins. In other words, it's as though the Contestant must act at time t independently of what happened prior to time t.

Having given a little intuition, let's elaborate by carefully phrasing the operation of the Game Show Test in language similar to that of the Game Show. We incorporate into the Game Show

Test A's random assignment to the primary points Z; we can imagine that the Host announces uniformly random values $A(z)$ for each $z \in Z$ that the Contestant must use. In the next step of the Game Show Test, we have the Contestant fix an odd Middle Function g; it's important to note that the Contestant gets to do this *after* seeing the random values $(A(z))_{z \in Z}$. Next we can imagine that the Host *secretly* picks a uniformly random primary point $\boldsymbol{z}_0 \in Z$ and a random half-path from it, $(\boldsymbol{z}_0, \boldsymbol{z}_1), \ldots, (\boldsymbol{z}_{\boldsymbol{T}-1}, \boldsymbol{z}_{\boldsymbol{T}})$. The Host will be testing equality of A's values on a random (according to w) edge along this half-path, using the value of g instead of A for the last point along the half-path. However in the Game Show *Test*, the Contestant must fix an entire strategy $(A(x))_{x \notin Z}$ before the Host reveals which edge is tested.

We'll now make things *easier* on the Contestant — this can only decrease the probability of the test rejecting. Specifically, we won't force the Contestant to immediately announce an entire strategy $(A(x))_{x \notin Z}$. Rather, the Host will reveal the edges $(\boldsymbol{z}_0, \boldsymbol{z}_1)$, $(\boldsymbol{z}_1, \boldsymbol{z}_2)$, ... one-by-one, just as in the Game Show, and will only require the Contestant to decide on $A(\boldsymbol{z}_t)$ in the tth step of this process. Note that $A(\boldsymbol{z}_0)$ is already fixed, as is $g(\boldsymbol{z}_{\boldsymbol{T}})$. Once $\boldsymbol{z}_{\boldsymbol{T}}$ is revealed, the Host will *then* choose $\boldsymbol{t}$ as in the Game Show Test and do the equality-test on the tth edge.

Now let's make a few more viewpoint changes, so that the Game Show Test becomes even more similar to the Game Show. As it has now been described, the Game Show Test starts at a random primary point $\boldsymbol{z}_0 \in Z$, and the Contestant is obliged to use the Host's initially randomly chosen value $A(\boldsymbol{z}_0)$. Let's change this so that once the Host chooses $\boldsymbol{z}_0 \in Z$ at random, the function g is immediately replaced by $\widetilde{g} = A(\boldsymbol{z}_0) g^{+\boldsymbol{z}_0} = g^{+A(\boldsymbol{z}_0)\boldsymbol{z}_0}$. (The last equality uses the fact that g is odd.) In this way, we may equivalently assume that (as in the Game Show) the random half-path always originates from $(1, 1, \ldots, 1)$, and that the Contestant is obliged to use the assignment $A(1, 1, \ldots, 1) = 1$. Now as the Game Show Test proceeds, when the Host reveals the tth edge, the Contestant is allowed to specify whether $A(\boldsymbol{z}_{\boldsymbol{t}})$ should be *equal* to $A(\boldsymbol{z}_{\boldsymbol{t}-1})$, or equal to its *negation*. Note that it is "costly" for the Contestant to choose negation, in the sense that it yields an "unequal" edge that will increase the Host's rejection probability by $w(t)$. This process proceeds until $\boldsymbol{z}_{\boldsymbol{T}}$ is reached, whereupon the Contestant is committed to the final assignment value $g(\boldsymbol{z}_{\boldsymbol{T}})$. In our final viewpoint change, instead of allowing the Contestant to either "keep" or "negate" the value assigned at time $t-1$, it is equivalent to allow the Contestant to either keep $\widetilde{g}$ or replace it by $-\widetilde{g}$. The latter choice corresponds to creating an "unequal" edge and incurring cost (rejection probability) $w(t)$. The fact that the Contestant is obliged to use the initial assignment 1 (recall we initially multiply $g^{+\boldsymbol{z}_0}$ by $A(\boldsymbol{z}_0)$) in the test corresponds to the fact that in the Game Show, the Contestant is obliged to end on 1.

It may seem as though we have by now shown that the Game Show Test is equivalent to playing the Game Show with Middle Function $g^{+A(\boldsymbol{z}_0)\boldsymbol{z}_0}$, where $\boldsymbol{z}_0 \in Z$ is chosen uniformly at random — in the sense that the rejection probability of the test is equal to the expected value of $\mathrm{cost}(g^{+A(\boldsymbol{z}_0)\boldsymbol{z}_0})$. However there is one catch: in the Game Show Test, g may be chosen *after* the random assignments $(A(z))_{z \in Z}$ to the primary points. The remainder of the proof is devoted to showing that the Contestant can not effectively "take advantage" of this.

For this we return to the actual Overall Gadget Test, which performs the Average Sensitivity Test with probability δ and the Game Show Test with probability $1 - \delta$. (Note that our theorem statement tolerates the loss of factor-$(1-\delta)$ in soundness here.) The purpose of the Average Sensitivity Test is to ensure that the chosen g must essentially be a junta. More precisely, we see that if the chosen g has average sensitivity exceeding $\frac{3/2}{\delta}$ then the Average Sensitivity component of the Overall Gadget Test already rejects with probability exceeding $\frac{3/2}{n}$. Thus we can assume the chosen g has average sensitivity at most $\frac{3/2}{\delta}$. It follows from Friedgut's Junta Theorem [Fri98] that g must be δ-close to a junta on some $J = 2^{O(1/\delta^3)}$ variables. Using the fact that g is odd, one

can also ensure that this junta is odd.[1] Next, note that the Game Show Test only involves g with probability $\frac{W(1/2)}{n}(1 \pm o_n(1)) \leq \frac{5}{n}$ (using Lemma 5.4). Furthermore, when it *does* involve g, the point on $\boldsymbol{x_T}$ on which it involves g is nearly uniformly random in $\{-1,1\}^n$. The difference from uniform randomness comes because we conditioned $\boldsymbol{T}$ to lie in the range $\frac{n}{2} \pm \sqrt{n \log n}$; however, $\boldsymbol{T}$ lies outside this range only with probability $o_n(1)$. Thus we see that if we simply replace g with its δ-close odd J-junta, we only affect the Overall Gadget Test's rejection probability by at most $\frac{5}{n}(\delta + o_n(1))$, an amount we can absorb in the theorem statement. In summary, we may freely assume that the chosen g is always an odd *junta* on $J = 2^{O(1/\delta^3)}$ coordinates.

Finally, to complete the proof by applying the Game Show Conjecture, it would remain to show that with probability $1 - o_n(1)$ over the initial random choice of $(A(z))_{z \in Z}$, for *all* choices of J out of n coordinates, the projection of the distribution $A(\boldsymbol{z}_0)\boldsymbol{z}_0$ to the coordinates J is pairwise independent and symmetric. This is not quite correct, but in Lemma 5.6 below we show that each projection is $O(\delta)$-close in total variation distance to being simultaneously pairwise-independent and symmetric. This is sufficient to complete the proof. □

Lemma 5.6. *Suppose n is sufficiently large as a function of δ. Suppose that for each of the $n/2$ strings $z \in Z$ a uniformly random bit $\boldsymbol{a}_z$ is chosen. Then except with probability $o_n(1)$, for every fixed set of $J = J(\delta)$ coordinates in $[n]$ (in particular, for $J = 2^{O(1/\delta^3)}$), the uniform distribution on the set of strings $(\boldsymbol{a}_z z_J)_{z \in Z}$ is $O(\delta)$-close (in fact, $o_n(1)$-close) to being simultaneously pairwise-independent and symmetric.*

Proof. We begin just by showing the pairwise-independence property; here we won't need that J is small compared to n. For any particular pair of coordinates (i,j) in $[n]$ the list of two-bit strings $(z_{\{i,j\}})_{z \in Z}$ has $n/4$ equal pairs and $n/4$ unequal pairs (since Walsh–Hadamard columns are orthogonal). Thus when the bits $\boldsymbol{a}_z$ are chosen, in the list $(\boldsymbol{a}_z z_{\{(i,j)\}})_{z \in Z}$ we will see each of the four possible two-bit strings $n/8 \pm O(\sqrt{n \log n})$ times except with probability $\ll 1/n^2$. Thus by taking a union bound over all pairs of coordinates (i,j), it follows that except with probability $o_n(1)$ we have that all projections of $(\boldsymbol{a}_z z)_{z \in Z}$ onto two coordinates are $\widetilde{O}(1/\sqrt{n})$-close to uniform.

Next we consider the symmetry property. Fix any subset $\mathcal{J} \subset [n]$ of J coordinates. Let's consider, for each string $x \in \{-1,1\}^J$, how many times it occurs in the list $(z_J)_{z \in Z}$. Some x's have at most $\sqrt{n}$ occurrences. However even collectively, these constitute only $2^J\sqrt{n}$ out of the $n/2$ strings, and thus they contribute only $o_n(1)$ total probability mass to the uniform distribution on $(z_J)_{z \in Z}$. As for the remaining strings $x \in \{-1,1\}^J$, each occurs at least $\sqrt{n}$ times. Thus when the bits $\boldsymbol{a}_z$ are chosen, nearly equally many occurrences of $\pm x$ will be formed. Taking into account the need to union-bound over all 2^J strings and indeed all $\binom{n}{J}$ subsets $\mathcal{J}$, we still get that except with probability $o_n(1)$, for all $\mathcal{J}$ of cardinality J, the projection of $(\boldsymbol{a}_z z)_{z \in Z}$ onto the J-coordinates is $O(\frac{\sqrt{J \log n}}{n^{1/4}})$-close to symmetric.

Finally, we claim that if a distribution on J-bit strings is $o_n(1)$-close to symmetric and has all 2-bit marginals $o_n(1)$-close to uniform, then it is has total variation distance at most $2J^2 \cdot o_n(1) = o_n(1)$ from being simultaneously symmetric and pairwise-independent. To see this, we can begin with the $o_n(1)$-nearby symmetric distribution; it still has all its 2-bit marginals at most $2o_n(1)$-close to uniform, and all of its 1-bit marginals are exactly uniform. Now we can apply the "correction procedure" from [AGM03] to deduce that the resulting distribution is $2J^2 o_n(1)$-close to being pairwise-independent. It only remains to observe that this correction procedure maintains symmetry, since it merely mixes the distribution with various symmetric distributions (namely, distributions of the form "uniform on $\{x : x_i x_j = 1\}$" for distinct $i, j \in \mathcal{J}$). □

[1] The proof shows g is close to $\mathrm{sgn}(\sum_{S \subseteq \mathcal{J}} \widehat{g}(S)\chi_S)$ for some $|\mathcal{J}| \leq J$, using an arbitrary convention for $\mathrm{sgn}(0)$. One need only ensure that any $\mathrm{sgn}(0)$ choices that arise are made in an oddness-preserving way.

This finishes the (Completeness) and (Soundness) cases of $\mathcal{G}$, giving us Theorem 5.3. □

5.3 Regarding the Game Show Conjecture

As mentioned, we are unable to prove the Game Show Conjecture. For the record, we describe here some of our ideas toward proving it. Since we are not claiming any theorems in this section, we will not be completely precise.

As we have already seen, it seems more natural to think of a "continuous" time parameter $u = t/n$ that starts at $u = 0$ and ends at $u = \frac{1}{2}$. (Thinking of n as large, the ending time will indeed be $u = \frac{1}{2} \pm o_n(1)$ with high probability.) In fact, we believe it's even more natural to use a different continuous parameterization of time. Specifically, define the *time remaining* parameter s by $s = \ln(2 - 2u)$; i.e., $u = 1 - \exp(s)/2$. As time runs from $u = 0$ up to $u = \frac{1}{2}$, the "time remaining" runs from $s = \ln 2$ down to $s = 0$. The idea behind this rescaling is that now the Host restricts coordinates according to an exponential clock of rate $\frac{1}{n}$. Note from (4) that the cost to the Contestant of negating $\widetilde{g}$ with s time remaining is

$$W(s) = 4\exp(-2s). \tag{5}$$

Suppose we have some current function $\widetilde{g}$ and the time remaining is s. Let us define $\mathrm{cost}_{\widetilde{g}}(s)$ to be the expectation of the Contestant's remaining cost, assuming an optimal strategy; note that $\mathrm{cost}(\widetilde{g}) = \mathrm{cost}_{\widetilde{g}}(\ln 2)$. For the two constant functions we have:

$$\mathrm{cost}_{+1}(s) = 0$$
$$\mathrm{cost}_{-1}(s) = 4\exp(-2s).$$

The latter equality holds because whenever the current function $\widetilde{g}$ gets restricted to the constant function -1, it is in the Contestant's best interest to immediately negate (because the negation cost is an increasing function of time).

As we alluded to earlier, for general $\widetilde{g}$ there is a "dynamic program" for computing $\mathrm{cost}_{\widetilde{g}}(s)$. (Actually, because s is a continuous parameter it's more like a "differential equation".) The "base case" for the dynamic program is

$$\mathrm{cost}_{\widetilde{g}}(0) = \begin{cases} 0 & \text{if } \widetilde{g}(1,1,\ldots,1) = 1, \\ 4 & \text{if } \widetilde{g}(1,1,\ldots,1) = -1. \end{cases} \tag{6}$$

In general, we can compute $\mathrm{cost}_{\widetilde{g}}(s + ds)$ given knowledge of $\mathrm{cost}_f(s)$ for all subfunctions f of $\pm\widetilde{g}$. The fact that the Contestant is allowed to actively negate $\widetilde{g}$ causes some complications, so let's begin by considering a *lazy* Contestant, meaning one who uses the (possibly nonoptimal) strategy of never negating $\widetilde{g}$ unless it gets restricted to the constantly -1 function (or unless the game ends and negation is "forced").

Writing $C_{\widetilde{g}}(s)$ for the analogue of $\mathrm{cost}_{\widetilde{g}}(s)$ in the case of a lazy Contestant, we have the same base case (6). As for the general formula, suppose that $\widetilde{g}$ depends on r coordinates and that we consider the time remaining dropping from $s+ds$ to s. Let $u = 1-\exp(s)/2$ as usual. It's easy to see that each of $\widetilde{g}$'s relevant coordinates has probability $\frac{du}{1-u}$ of being restricted. Since $u = 1-\exp(s)/2$, this precisely equals $-ds$, and hence in going from $s + ds$ to s time remaining, each coordinate has probability ds of being restricted. Thus we deduce the "dynamic programming" formula:

$$C_{\widetilde{g}}(s + ds) = r \cdot ds \cdot \operatorname*{avg}_{\widetilde{g}'}\{C_{\widetilde{g}'}(s)\} + (1 - r \cdot ds) \cdot C_{\widetilde{g}}(s),$$

where the average is over all functions $\widetilde{g}'$ gotten by restricting one coordinate of $\widetilde{g}$ to be -1. Let's write

$$A_{\widetilde{g}}(s) = \operatorname*{avg}_{\widetilde{g}'}\{C_{\widetilde{g}'}(s)\},$$

and also approximate $C_{\widetilde{g}}(s+ds) = C_{\widetilde{g}}(s) + \frac{d}{ds}C_{\widetilde{g}}(s)$. Then after rearrangement, the above dynamic programming formula becomes the differential equation

$$-C_{\widetilde{g}}(s) = r(C_{\widetilde{g}}(s) - A_{\widetilde{g}}(s)).$$

The solution to this ODE is

$$C_{\widetilde{g}}(s) = r\exp(-rs)\cdot\mathcal{L}_r A_{\widetilde{g}}(s), \tag{7}$$

where $\mathcal{L}_r$ denotes the operator defined by

$$\mathcal{L}_r A(s) = \int_0^s \exp(ry)A(y)\,dy + \text{const},$$

with the value of const being determined by the initial condition, $C_{\widetilde{g}}(0) \in \{0, 4\}$. With these formulas in hand one can directly compute the following formulas for the two $r = 1$ functions:

$$\begin{aligned} C_{d_i}(s) &= 4\exp(-s) - 4\exp(-2s), \\ C_{-d_i}(s) &= 4\exp(-s). \end{aligned}$$

It was previously argued that the lazy strategy is optimal when the Middle Function is a dictator d_i, and the first formula above confirms that $\text{cost}(d_i) = 4\exp(-\ln 2) - 4\exp(-2\ln 2) = 1$. Furthermore, observe that

$$C_{-d_i}(s) = C_{d_i}(s) + W(s).$$

The left-hand side is the expected cost to the Contestant when using the lazy strategy on a negated dictator; the right-hand side is the expected cost if the Contestant decides to negate $-d_i$ to d_i at time s. Notice that we have equality for all s. This is precisely by design; as mentioned earlier, we chose the negation-cost formula $W(s)$ so that when the Middle Function is a negated dictator, all strategies of the form "negate if and only the relevant restriction has not occurred by time s_0" are equally good, including the lazy strategy.

We have $\text{cost}(d_i) = 1$ and $\text{cost}(-d_i) = 4\exp(-\ln 2) = 2$, hence $\mathbf{E}_{\boldsymbol{b}\sim\mathcal{D}}[\text{cost}(g^{+\boldsymbol{b}})] = \frac{3}{2}$ whenever $\mathcal{D}$ is symmetric and g is a dictator or negated dictator. To confirm the Game Show Conjecture, what we need to show is that for $\mathcal{D}$ symmetric and pairwise-independent we have

$$\mathop{\mathbf{E}}_{\boldsymbol{b}\sim\mathcal{D}}[\text{cost}(g^{+\boldsymbol{b}})] \geq \frac{3}{2}$$

for all odd g.

Next we consider functions with exactly 2 relevant variables. There are actually no such *odd* functions, but we still need to analyze them since they can arise at intermediate points in the game. From (7) one may compute:

$$C_{\text{AND}_2}(s) = 8(\exp(s) - 1 - s)\exp(-2s) \qquad C_{\text{OR}_2}(s) = 8s\exp(-2s) \tag{8}$$

$$C_{\text{NOR}_2}(s) = 4\exp(-2s) \qquad C_{=_2}(s) = 4(2\exp(s) - 1 - 2s)\exp(-2s) \tag{9}$$

$$C_{\neq_2}(s) = 8(\exp(s) - 1)\exp(-2s) \qquad C_{x\wedge\neg y}(s) = 4(\exp(s) - 1)\exp(-2s) \tag{10}$$

$$C_{x\vee\neg y}(s) = 4\exp(s)\exp(-2s) \qquad C_{\text{NAND}_2}(s) = 4(2\exp(s) - 1)\exp(-2s). \tag{11}$$

Note that $C_{\mathrm{NAND}_2}(s) > W(s) + C_{\mathrm{AND}_2}(s)$ for all $s > 0$. This implies that the lazy strategy is not optimal for the Contestant when the function is NAND_2. It is not hard to show that the best strategy for the Contestant involves immediately negating NAND_2 to AND_2 whenever it arises. For all other functions above the lazy strategy is optimal, i.e., $\mathrm{cost}_g(s) = C_g(s)$; but for NAND_2 we have

$$\mathrm{cost}_{\mathrm{NAND}_2}(s) = W(s) + \mathrm{cost}_{\mathrm{AND}_2}(s) = 4(2\exp(s) - 1 - 2s)\exp(-2s).$$

Now we can consider 3-bit functions. The only odd ones (up to symmetry) are $\chi_{\{1,2,3\}}$ and Maj_3^{+b} for $b \in \{-1,1\}^3$. It's an exercise to confirm the Game Show Conjecture for all parity functions, so let's focus on the majority-type functions. For them we'll introduce the more general notation $\mathrm{LTF}_{a_1,\dots,a_r}(x) = \mathrm{sgn}(a_1x_1 + \cdots + a_rx_r)$. From (7) one may compute:

$$\begin{aligned}
C_{\mathrm{LTF}_{1,1,1}}(s) &= 24(1 - \exp(-s) + s\exp(s))\exp(-3s)\\
C_{\mathrm{LTF}_{1,1,-1}}(s) &= 8(\exp(2s) - s\exp(s) - 1)\exp(-3s)\\
C_{\mathrm{LTF}_{1,-1,-1}}(s) &= 4(2\exp(s) - 2s - 1)\exp(-2s)\\
C_{\mathrm{LTF}_{-1,-1,-1}}(s) &= 4(3\exp(s) - 2)\exp(-3s).
\end{aligned}$$

One can show that in fact cost $= C$ for the first three of these functions. However this is not true for the last function, namely $-\mathrm{Maj}_3$. Here we have $C_{-\mathrm{Maj}_3}(s) > C_{\mathrm{Maj}_3}(s) + W(s)$, implying that sometimes the Contestant should negate $-\mathrm{Maj}_3$ to Maj_3. Indeed, now one should consider the overall "dynamic program" more carefully, to incorporate the fact that the Contestant is allowed to negate. Extending the differential equation reasoning above, one finds that it is optimal for the Contestant to negate $\widetilde{g}$ at time s_0 if and only if

$$A_{\widetilde{g}}(s_0) \le A_{-\widetilde{g}}(s_0) + W(s_0) + W'(s_0)/r.$$

In particular, when $\widetilde{g} = -\mathrm{Maj}_3$, this condition is equivalent to

$$\mathrm{cost}_{\mathrm{NOR}_2}(s_0) \le \mathrm{cost}_{\mathrm{OR}_2}(s_0) + W(s_0)/3.$$

Using the formulas in (8), one directly calculates that the least s_0 for which we have equality above is $s_0 = 1/3$. It follows that the optimal strategy for $-\mathrm{Maj}_3$ is to negate if and only if the time remaining reaches $1/3$ without any relevant coordinates being restricted. Taking this into account, we ultimately determine:

$$\mathrm{cost}_{-\mathrm{Maj}_3}(s) = \begin{cases} 24(1 - \exp(-s) + s\exp(s))\exp(-3s) + 4\exp(-2s) & \text{if } s \le 1/3,\\ 12(2 + \exp(s) - 2\exp(1/3))\exp(-3s) & \text{if } s \ge 1/3. \end{cases}$$

By substituting $s = \ln 2$, we can state the optimal costs for all reorientations of Maj_3:

$$\begin{aligned}
\mathrm{cost}(\mathrm{LTF}_{1,1,1}) &= 6\ln 2 - 3 \approx 1.159\\
\mathrm{cost}(\mathrm{LTF}_{1,1,-1}) &= 3 - 2\ln 2 \approx 1.614\\
\mathrm{cost}(\mathrm{LTF}_{1,-1,-1}) &= 3 - 2\ln 2 \approx 1.614\\
\mathrm{cost}(\mathrm{LTF}_{-1,-1,-1}) &= 6 - 3\exp(1/3) \approx 1.813.
\end{aligned}$$

The last of these, $\mathrm{cost}(-\mathrm{Maj}_3)$, is "surprisingly low". In particular, note that

$$\mathrm{avg}\{\mathrm{cost}(\mathrm{Maj}_3), \mathrm{cost}(-\mathrm{Maj}_3)\} = \frac{3}{2}(1 - \exp(1/3) + 2\ln 2) \approx .99 \cdot \frac{3}{2}. \tag{12}$$

This shows that it is *not* sufficient in the Game Show Conjecture merely to assume that the distribution on orientations $\boldsymbol{b}$ is symmetric. However, as the only symmetric pairwise-independent distribution on three bits is the uniform distribution, the case of $g = \mathrm{Maj}_3$ is consistent with the Game Show Conjecture:

$$\tfrac{1}{8}(6\ln 2 - 3) + \tfrac{3}{8}(3 - 2\ln 2) + \tfrac{3}{8}(3 - 2\ln 2) + \tfrac{1}{8}(6 - 3\exp(1/3)) \approx 1.58 > \tfrac{3}{2}.$$

We do not have a clear strategy for analyzing the general case. A reasonable place to start is to analyze all linear threshold functions, a class of functions closed under restriction, which includes dictators and majorities. In particular, the r-ary "monarchy" function, $\mathrm{LTF}_{r-2,1,1,\dots,1}$ seems like an interesting challenge to analyze.

We make one final remark: As we have seen, the Game Show Conjecture is not correct if the distribution on orientations $\boldsymbol{b}$ need only be symmetric. This is a bit of a shame, because in the proof of the soundness Theorem 5.5, one can ensure symmetry without using any special properties of the predicate being reduced from, besides uselenessness. In particular, one could use the older NP-hardness reduction of Samorodnitsky and Trevisan [ST00] in place of Chan's, which has the advantage [MR10] of holding with a quasilinear-size blowup. If one hazards the guess that $\mathbf{E}_{\boldsymbol{b}\sim\mathcal{D}}[\mathrm{cost}(g^{+\boldsymbol{b}})] \geq .99 \cdot \frac{3}{2}$ whenever $\mathcal{D}$ is symmetric, a consequence would be that 2-Lin(2) is NP-hard to approximate to factor $.99 \cdot \frac{3}{2}$ with a quasilinear-size reduction; hence, this level of approximation would require nearly full exponential time, $2^{n^{1-o(1)}}$, assuming the Exponential Time Hypothesis.

6 Limitations of gadget reductions

In this section, we show a limitation to proving inapproximability using gadget reductions from balanced pairwise-independent predicates: that is, predicates ϕ that admit a set $S \subseteq \mathsf{sat}(\phi)$ satisfying Property 2 in Definition 2.4. We show that gadget reductions from ϕ to 2-Lin(2) can not prove inapproximability larger than a factor-2.54 for the deletion version. Note that this applies to the Had_k predicates and to a broader class of predicates that do not necessarily admit a natural group operation.

Theorem 6.1. *Let $\mathcal{G}$ be a (c,s)-generic gadget reducing Max-ϕ to Max-2-Lin(2), where ϕ admits a balanced pairwise-independent set. Then*

$$\frac{s}{c} \leq \frac{1}{1 - e^{-1/2}} \approx 2.54.$$

Proof. As before, K is the number of satisfying assignments of ϕ. Recall that the vertex set of $\mathcal{G}$ is $V = \{-1,1\}^K$. Further, via Propositions 2.19 and 2.20, we need only consider folded assignments to these variables, and we can assume $\mathcal{G}$ only uses (=)-constraints. Finally, via Proposition 2.27, we can assume that every (=)-constraint used by $\mathcal{G}$ is between two variables x and y which are Hamming distance one from each other. Let P be the set of generic primary variables, let $-P$ be their negations, and let $P^{\pm} = P \cup (-P)$ denote the union of the two. Since ϕ is balanced pairwise-independent, we have a set $S \subseteq [K]$ so that for i picked uniformly at random from S, $\mathbf{Pr}_i[u_i = v_i] = 1/2$ for distinct primary variables $u, v \in P$.

Define the similarity between x and y to be $\mathsf{sim}(x,y) := \mathbf{Pr}_i[x_i = y_i]$ and set $\mathsf{sim}(x, P^{\pm}) := \max_{y \in P^{\pm}} \mathsf{sim}(x,y)$. Pairwise-independence allows us to claim that any variable x is strongly similar (i.e. has similarity $> \frac{3}{4}$) with at most one variable $y \in P^{\pm}$; define y to be x's *closest* primary variable.

Fact 6.2. *For any $x \in V$, if $\mathsf{sim}(x, y) > \frac{3}{4}$ for some $y \in P^{\pm}$, then $\mathsf{sim}(x, y') < \frac{3}{4}$ for all other $y' \in P^{\pm}$.*

Proof. If x has $\mathsf{sim}(x, y_1) > \frac{3}{4}$ and $\mathsf{sim}(x, y_2) \geq \frac{3}{4}$ for $y_1, y_2 \in P^{\pm}$, then

$$\mathsf{sim}(y_1, y_2) \geq \mathsf{sim}(y_1, x) + \mathsf{sim}(x, y_2) - 1 > \frac{1}{2},$$

contradicting the assumption on ϕ. □

This fact allows us to design the following "threshold-rounding" procedure to construct a distribution $\mathcal{A} \in \mathsf{R}^{gen}(\phi)$. Let $\mathcal{D}$ be a distribution over $[3/4, 1]$ with probability density function $\mathcal{D}(t) = C \cdot e^{2t}$, for $t \in [3/4, 1]$ (and C set appropriately).

1. Pick a random assignment to the primary variables.
2. Pick a number $t \sim \mathcal{D}$. For any variable $x \in V$, call x type 1 if $\mathsf{sim}(x, P^{\pm}) > t$ and type 2 otherwise.
3. Assign all type-1 variables the value of their closest primary variable.
4. Pick a uniformly random dictator d_i and set all the type-2 variables to agree with this dictator.
5. Output the resulting assignment.

Note that the assignments are folded and are random on the primary variables. We analyse the performance of this assignment. Let (x, y) be an edge in $\{-1, 1\}^K$ of Hamming weight one. If both $\mathsf{sim}(x, P^{\pm}), \mathsf{sim}(y, P^{\pm}) \leq \frac{3}{4}$, then regardless of the value of t, x and y will both always be type-2 variables, in which case $\mathcal{A}$ violates the edge between them with the probability of a random dictator, which is $\frac{1}{K} \leq \frac{1}{1-e^{-1/2}} \cdot \frac{1}{K}$.

On the other hand, suppose WLOG that $\mathsf{sim}(x, P^{\pm}) > \mathsf{sim}(y, P^{\pm})$ and that $\mathsf{sim}(x, P^{\pm}) > \frac{3}{4}$. If we set $s := \mathsf{sim}(y, P^{\pm})$, then $\mathsf{sim}(x, P^{\pm}) = s + \frac{1}{K}$. Because y is distance one from x, $s \geq \frac{3}{4}$. Not only that, if y has a closest primary variable, then that variable is the same as x's closest primary variable (this is by Fact 6.2). Now, to calculate the probability that $\mathcal{A}$ violates (x, y), there are three cases:

1. If $t \in \left[\frac{3}{4}, s\right)$, then x and y are assigned the value of the same variable in $P^{\pm}$, so (x, y) is never violated in this case.
2. If $t \in \left[s, s + \frac{1}{K}\right)$, then y's value is chosen according to a uniformly random dictator assignment, meaning that it is a uniformly random ± 1-bit. independent from x's value In this case, (x, y) is violated with probabiltiy $\frac{1}{2}$.
3. If $t \in \left[s + \frac{1}{K}, 1\right]$, then both x and y are assigned values according to a random dictator, in which case (x, y) is violated with probability $\frac{1}{K}$.

In total,

$$\begin{aligned}\mathbf{Pr}[\mathcal{A} \text{ violates } (x,y)] &= \frac{1}{2} \cdot \mathop{\mathbf{Pr}}_{t\sim\mathcal{D}}[t \in [s, s+1/K)] + \frac{1}{K} \cdot \mathop{\mathbf{Pr}}_{t\sim\mathcal{D}}[t \in [s+1/K, 1)] \\ &= \frac{1}{2}\int_s^{s+\frac{1}{K}} Ce^{2t}\mathrm{d}t + \frac{1}{K}\int_{s+\frac{1}{K}}^1 Ce^{2t}\mathrm{d}t \\ &\le \frac{1}{2} \cdot \frac{Ce^{2s+2/K}}{K} + \frac{1}{K}\int_{s+\frac{1}{K}}^1 Ce^{2t}\mathrm{d}t \\ &= \frac{Ce^2}{2K} = \frac{1}{1-e^{-1/2}} \cdot \frac{1}{K},\end{aligned}$$

as promised. Here the inequality follows from the fact that e^{2t} is an increasing function. As $\mathcal{G}$ only uses length-one edges, $c = \frac{1}{K}$. We have just shown that $\mathsf{uval}(\mathcal{A};\mathcal{G}) \le \frac{1}{1-e^{-1/2}} \cdot \frac{1}{K}$. Because $\mathcal{A} \in \mathsf{R}^{gen}(\phi)$, we conclude that $\frac{s}{c} \le \frac{1}{1-e^{-1/2}}$. □

7 Conclusion

As mentioned, we view our factor-$\frac{11}{8}$ NP-hardness result more as a proof of concept, illustrating that the longstanding barrier of factor-$\frac{5}{4}$ NP-hardness for Max-Cut/2-Lin(2)/Unique-Games can be broken. There are quite a few avenues for further work:

- An obvious problem is to derive a better NP-hardness result for 2-Lin(2) by reduction from Had_4 rather than Had_3. As one can always embed our Had_3-based gadget into a standard Had_4-based gadget, this method will always yield a hardness of at least $\frac{11}{8}$. But presumably the optimal Had_4-based gadget will do slightly better.

- Since our analysis of the optimal Had_3 gadget is already somewhat complicated, it might be challenging to analyze the Had_4 case explicitly. A weaker but more plausible goal would be to prove (perhaps indirectly) that there *exists* a $\delta_0 > 0$ such that the optimal Had_4 gadget achieves factor-$(\frac{11}{8} + \delta_0)$ NP-hardness. This would at least definitely establish that $\frac{11}{8}$ is not the "correct answer" either.

- Of course, proving the Game Show Conjecture would yield the improved NP-hardness factor of $\frac{3}{2}$. It may also be simpler to try to prove a non-optimal version of the conjecture, yielding *some* hardness factor better than $\frac{11}{8}$ but worse than $\frac{3}{2}$. Certain ideas we had for trying to prove the conjecture (e.g., distinguishing whether g is "close to" or "far from" being a dictator) might yield such a result.

- Along these lines, it could also be a good idea to prove some form of the Game Show Conjecture that only relies on the distribution $\mathcal{D}$ on orientations $\boldsymbol{b}$ being symmetric, rather than pairwise-independent. As mentioned, this would allow one to reduce from the Samorodnitsky–Trevisan hardness result, yielding nearly full-exponential hardness under the ETH. Since our proof of Theorem 5.2 reduces from Chan's hardness result, it has the unfortunate aspect that we only get factor-$(\frac{3}{2} - \delta)$ hardness (under ETH) for algorithms running in time $2^{n^{\delta'}}$ for some $\delta' = \delta'(\delta)$ that we did not bother to make explicit.

- For Max-Cut, our work establishes NP-hardness of $(\epsilon, C\epsilon)$-approximation for any $C < \frac{11}{8}$, but only for $\epsilon \le \epsilon_0$ where ϵ_0 is some not-very-large constant arising out of Proposition 2.36. It

would be nice to get a direct Max-Cut gadget yielding a larger ϵ_0, like the $\epsilon_0 = \frac{1}{8}$ we have for 2-Lin(2).

- A recent result of Gupta, Talwar, and Witmer [GTW13] showed NP-hardness of approximating the (closely related) Non-Uniform Sparsest Cut problem to factor-$\frac{17}{16}$, by designing a gadget reduction from the old $(\frac{4}{21}, \frac{5}{21})$-approximation hardness of Håstad [Hås97]. A natural question is whether one can use ideas from this paper to make a direct reduction from Had_2 or Had_3 to Non-Uniform Sparsest Cut, improving the NP-hardness factor of $\frac{17}{16}$.

- We are now in the situation (similar to the situation prior to [OW12]) wherein the best NP-hardness factor we know how to achieve for 2-Lin(q) (or Unique-Games) is achieved by taking $q = 2$. In fact, we don't know how to achieve an NP-hardness factor better than $\frac{5}{4}$ for 2-Lin(q) for any $q > 2$, even though 2-Lin(q) is presumably *harder* for larger q. Can this situation be remedied?

- In light of the limitations described in Section 6, it makes sense to seek alternative methodology of establishing improved NP-hardness for 2-CSPs. An example showing that this is not at all hopeless comes from the decade-old work of Chlebík and Chlebíková [CC04], which shows NP-hardness of approximating 2-Sat(-Deletion) to factor $8\sqrt{5} - 15 \approx 2.8885$. Their result is essentially a small tweak to the Vertex-Cover hardness of Dinur and Safra [DS02] and thus indeed uses a fairly radical methodology for establishing two-bit CSP-hardness, namely direct reduction from a specialized Label-Cover-type problem.

Acknowledgments

The authors would like to warmly thank Per Austrin for his assistance with computer analysis of the $\frac{11}{8}$-gadget.

References

[ABS10] Sanjeev Arora, Boaz Barak, and David Steurer. Subexponential algorithms for Unique Games and related problems. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science*, pages 563–572, 2010. 1.5

[ACMM05] Amit Agarwal, Moses Charikar, Konstantin Makarychev, and Yury Makarychev. $O(\sqrt{\log n})$ approximation algorithms for Min-Uncut, Min-2CNF-Deletion, and directed cut problems. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 573–581, 2005. 1.2

[AGM03] Noga Alon, Oded Goldreich, and Yisha Mansour. Almost k-wise independence versus k-wise independence. *Information Processing Letters*, 88(3):107–110, 2003. 5.2

[AH12] Per Austrin and Johan Håstad. On the usefulness of predicates. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity*, pages 53–63, 2012. 1.2, 2

[ARV04] Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings and graph partitioning. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 222–231, 2004. 1.1

[BGS95] Mihir Bellare, Oded Goldreich, and Madhu Sudan. Free bits, PCPs, and non-approximability – towards tight results. Technical Report TR95-024, Electronic Colloquium on Computational Complexity, 1995. 1.1

[CC04] Miroslav Chlebík and Janka Chlebíková. On approximation hardness of the minimum 2SAT-DELETION problem. In *Proceedings of the 29th Annual International Symposium on Mathematical Foundations of Computer Science*, pages 263–273, 2004. 7

[Cha13] Siu On Chan. Approximation resistance from pairwise independent subgroups. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing*, pages 447–456, 2013. 1.2, 2, 2.12, 2.1

[CMM06] Moses Charikar, Konstantin Makarychev, and Yury Makarychev. Near-optimal algorithms for Unique Games. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pages 205–214, 2006. 1.3

[DS02] Irit Dinur and Samuel Safra. The importance of being biased. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 33–42, 2002. 7

[FR04] Uriel Feige and Daniel Reichman. On systems of linear equations with two variables per equation. In *Proceedings of the 7th Annual International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 117–127, 2004. 1

[Fri98] Ehud Friedgut. Boolean functions with low average sensitivity depend on few coordinates. *Combinatorica*, 18(1):27–36, 1998. 5.2

[GTW13] Anupam Gupta, Kunal Talwar, and David Witmer. Sparsest Cut on bounded treewidth graphs: algorithms and hardness results. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing*, pages 281–290, 2013. 7

[GW94] Michel Goemans and David Williamson. A 0.878 approximation algorithm for MAX-2SAT and MAX-CUT. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 422–431, 1994. 1.1

[Hås97] Johan Håstad. Some optimal inapproximability results. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 1–10, 1997. 1.1, 1.6, 1.1, 1.2, 7

[Kho02] Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 767–775, 2002. 1, 1.1

[KKMO07] Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for Max-Cut and other 2-variable CSPs? *SIAM Journal on Computing*, 37(1):319–357, 2007. 1, 1.4

[MOO10] Elchanan Mossel, Ryan O'Donnell, and Krzysztof Oleszkiewicz. Noise stability of functions with low influences: invariance and optimality. *Annals of Mathematics*, 171(1):295–341, 2010. 1, 1.4

[MR10] Dana Moshkovitz and Ran Raz. Two-query PCP with subconstant error. *Journal of the ACM*, 57(5):29, 2010. 1.1, 1.6, 5.3

[OW12] Ryan O'Donnell and John Wright. A new point of NP-hardness for Unique-Games. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing*, pages 289–306, 2012. 1.1, 7

[Rao11] Anup Rao. Parallel repetition in projection games and a concentration bound. *SIAM Journal of Computing*, 40(6):1871–1891, 2011. 1.4

[ST00] Alex Samorodnitsky and Luca Trevisan. A PCP characterization of NP with optimal amortized query complexity. In *Proc. 32nd ACM Symposium on Theory of Computing*, pages 191–199, 2000. 5.3

[TSSW00] Luca Trevisan, Gregory Sorkin, Madhu Sudan, and David Williamson. Gadgets, approximation, and linear programming. *SIAM Journal on Computing*, 29(6):2074–2097, 2000. 1.1, 1.2, 2.1, 2.1, 2.1, 2.5

[Wil05] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2-3):357–365, 2005. 1.1