Electronic Colloquium on Computational Complexity, Report No. 83 (2014)

The Computational Benefit of Correlated Instances

Irit Dinur
Weizmann Institute of Science
irit.dinur@weizmann.ac.il

Shafi Goldwasser
MIT and Weizmann Institute of Science
shafi@theory.csail.mit.edu

Huijia Lin
University of California, Santa Barbara
rachel.lin@cs.ucsb.edu

Abstract

The starting point of this paper is that instances of computational problems often do not exist in isolation. Rather, multiple and correlated instances of the same problem arise naturally in the real world. The *challenge* is how to gain computationally from instance correlations when they exist. We will be interested in settings where significant computational gain can be made in solving a single primary instance by having access to additional auxiliary instances which are correlated to the primary instance via the solution space.

We focus on Constraint Satisfaction Problems (CSPs), a very expressive class of computational problems that is well-studied both in terms of approximation algorithms and NP-hardness and in terms of average case hardness and usage for cryptography, e.g. Feige's random 3-SAT hypothesis, Goldreich's one way function proposal, learning-parity-with-noise, and others.

To model correlations between instances, we consider *generating processes* over search problems, where a primary instance I is first selected according to some distribution D (e.g. worst case, uniform, etc); then auxiliary instances $I_1, ..., I_T$ are generated so that their underlying solutions $S_1, ..., S_T$ each are a "perturbation" of a primary solution S for I. For example, S_t may be obtained by the probabilistic process of flipping each bit of S with a small constant probability.

We consider a variety of naturally occurring worst case and average case CSPs, and show how availability of a small number of auxiliary instances generated through a natural generating process, radically changes the complexity of solving the primary instance, from intractable to expected polynomial time. Indeed, at a high-level, knowing a logarithmic number of auxiliary instances enables a close polynomial time approximation of the primary solution, and when in addition the "difference vector" between the primary and the auxiliary solution is known, the primary solution can be exactly found. Furthermore, knowing even a single auxiliary instance already enables finding the exact primary solution for a large class of CSPs.

ISSN 1433-8092

1 Introduction

The integer factorization problem has fascinated mathematicians for centuries and computer scientists for decades. It has helped us illucidate some of the key concepts at the heart of the theory of computation, providing arguably the most elegant example of an NP problem which is hard to solve but easy to verify by grade school mathematics, the basis for the first public key encryption scheme, and the impetus for much of modern day research into the power of quantum over classical computation.

Recently, [HDWH12] pointed out a simple but pervasive problem with using integer factorization as a basis for public-key cryptography. Many composite and hard to factor numbers used in practice, are obtained by first generating their prime divisors using the outputs of weak pseudo random number generators with correlated seeds. As a result, one can obtain *multiple instances* of composite numbers with correlated prime divisors, which render the composite number instances trivial to factor. At the extreme, two instances $n = pq$ and $n' = pq'$ share a prime factor. It is trivial to then factor n and n' by computing their greatest common divisor.

This example of problem instances which are hard when they stand alone, but are easy when given correlated instances, may seem anecdotal. Our thesis is that the situation may be quite to the contrary. Instances of computational problems with correlated solutions seem to arise naturally in many scenarios. *The challenge is not to find settings which present correlated instances, but how to take advantage of correlations when they exist.*

To mention a few "real world" examples. In biology, the problem of learning the mapping from DNA (genotype) to the observable characteristics (phenotype) of an organism, draws on data from multiple specimens with highly correlated DNA providing multiple correlated inputs. In coding theory, the objective is to recover from as many possible errors as possible: to this end, one may well take advantage of the fact that a typical hard-drive contains error-correcting encodings of multiple files that are within small edit-distance from each other. In learning theory, one well-studied goal is to classify future examples from known examples of a hidden concept class. It is likely, that many concepts to be learned are correlated, and access to examples from correlated concepts can reduce the number of examples necessary to learn any one of them.

In this paper, we focus on the complexity of constraint satisfaction problems (CSPs) with access to multiple instances with correlated solutions. The class of constraint satisfaction problems is a very rich and well-studied class of computational problems. This class includes many NP-hard problems (such as 3SAT, max-cut, 3-coloring) whose precise approximation behavior has been extensively studied. It is also believed that random instances of CSPs (under an appropriately defined distribution) are hard and this has been the content of several influential conjectures including Goldreich's one way function proposal [Gol00], Feige's random 3SAT hypothesis [Fei02], learning parity with noise hypothesis [McE78, BFKL93, Ale03], and various extensions and follow up works e.g. [ABW10, BKS13, DLSS13]. We show that in various settings, hard CSP instances become easy when correlated instances are available.

We view this work as a first step in a wider study of how to exploit correlations in available inputs.

Correlation. To model correlation between problem instances, we think of a *generating process* over search problems: In the process, a primary instance I (e.g., a CSP instance) with an underlying hidden solution S (e.g., an assignment satisfying all constraints) is selected according to some distribution Pri (e.g. worst case, uniform, etc.). Next, some auxiliary instances $\{I_1, I_2, \ldots\}$ are selected according to another distribution $\mathsf{Aux}(I, S)$ depending on the primary instance I and

solution S. The algorithmic goal is to recover a solution $\tilde{S}$ to the primary instance I, given additionally the collection of auxiliary instances (but not their solutions).

What types of correlation between instances are interesting to study? We first observe that if the auxiliary instances can be sampled efficiently given the primary instance but not the solution (that is, $I_j \xleftarrow{\$} \mathsf{Aux}(I)$ can be sampled efficiently); then access to such auxiliary instances can at most lead to a polynomial-time speedup. In this work, we are interested in qualitative computational gains so we naturally consider correlation based also on the *solution* of the primary instance. Two natural and interesting models are the *randomly-perturbed-solution model* and the *known-differences model*. Below we explain the two models in the context of CSP.

CSPs. Formally, a constraint satisfaction problem (CSP) is parameterized by a predicate $P : \{0,1\}^d \to \{0,1\}$, and a density parameter $D \geq 1$. An instance of this problem on n input variables $x = x_1, \ldots x_n$ is a set of $m = nD$ constraints specified by a bi-partite graph $G = ([n], [m], E)$ and a string $y \in \{0,1\}^m$. The graph connects each "output vertex" y_k (for $k = 1, \ldots, m$) to d "input vertices" which we index by $G(k,1), G(k,2), \ldots, G(k,d)$. Each output vertex specifies an equation constraint on the input variables,

$$\forall k \in [m], \quad P(x_{G(k,1)}, \ldots, x_{G(k,d)}) = y_k$$

(see Section 4.1 for detailed definition). Given a CSP instance, represented succinctly as a pair (G, y), the goal is to find a solution, i.e., an assignment $\tilde{x}$ maximizing the fraction of satisfied constraints. Throughout this paper, we focus on CSPs whose solution satisfies all of the constraints.

This definition of a CSP is "functional" in the sense that the graph G and the predicate P together define a function mapping inputs x to outputs y, such that the CSP problem is to invert this function on a given output string y. This differs from the more standard definition of a P-CSP (for example think of 3-SAT) where the instance is a collection of d-tuples of variables on which the predicate P should evaluate always to *true*. The advantage in our functional definition is that it allows efficient sampling of a pair of instance and solution while at the time remaining to be (conjectured) hard. In contrast, sampling satisfiable 3-SAT formulae, for example, is a process that is notoriously hard to analyze[1].

Correlated CSPs. In the randomly-perturbed-solution model, the underlying solutions of the auxiliary instances are "random perturbations" of the primary solution. In the context of CSP, in addition to a primary instance (G, y), multiple *auxiliary* instances $(G^1, y^1), \ldots (G^T, y^T)$ correlated with (G, y) are generated according to the following process $Gen_{\mathsf{Pri},\mathsf{Aux}}$:

> On input n, T, ε where N is the length of the primary instance, T the number of auxiliary instances, and $\varepsilon \in (0, 1/2]$ the perturbation parameter, do:
>
> 1. Choose a primary instance and solution (G, y, x) from the distribution Pri.
> 2. Perturb the solution: For each $t = 1, 2, \ldots, T$, let x^t be an ε-noisy copy of x derived by flipping each bit of x with probability ε independently.

[1]There are two average case distributions for satisfiable SAT formulae that are studied in the literature: "conditional" and "planted". In the "conditional" distribution one samples a random k-SAT formula with given number of clauses and variables, conditioned on it being satisfiable. In the "planted" distribution one chooses a random solution, and then samples satisfied clauses independently at random. The conditional distribution is notoriously hard to analyze, so instead significant effort has been devoted to the planted distribution, resulting in polynomial time algorithms for example for finding a solution in a 3-SAT instance with large enough constant clause-to-variable ratio [Fla08]. This means that even without correlated inputs, the problem is easy; and this directs our efforts elsewhere.

3. Choose T auxiliary instances: For each $t = 1, 2, \ldots, T$, choose an instance (G, y^t) from the distribution $\mathsf{Aux}(x^t)$ of instances with solution x^t.

Output $(G, y), (G, y^1), (G, y^2), \ldots, (G, y^T)$.

Observe that in our model all instances have the exact same G. The distributions Pri and $\mathsf{Aux}(x^t)$ specify how the primary instance and auxiliary instance with solution x_t are chosen in Step 1 and 3. Naturally, different instantiations of these distributions lead to different variants of the randomly-perturbed-solution model. For instance, Pri could be choosing a worst-case CSP instance or a random one, leading to the question of solving worst-case or random CSP, given auxiliary instances[2]. The algorithmic goal is to recover a solution $\tilde{x}$ for (G, y), given $(G, y), (G, y^1), (G, y^2), \ldots, (G, y^T)$.

A tampering-oriented[3] model of correlated instances would allow the algorithm itself to control the perturbation process (by algorithm we always mean the algorithm whose goal is to recover a solution to the primary instance (G, y)). This however may give undue power to the algorithm designer, and make results less interesting. We instead consider a second model which can be considered as an intermediate model between tampering and the randomly-perturbed-solution model above. In this model the perturbation is still performed randomly, but the algorithm is given, in addition to the auxiliary instance, the exact coordinates in which bits were flipped although not the values of the bits. Formally, the algorithm is given $\{(G, y_t)\}_t$ and $\{\Delta_t = x_t \oplus x\}_t$ generated by $Gen_{\mathsf{Pri},\mathsf{Aux}}$. The algorithmic task is easier in this model compared to the randomly-perturbed-solution model and harder than the tampering model. This model turns out to be technically interesting and useful because it pinpoints an intermediate step in a natural class of algorithms that search for a solution $\tilde{x}$ by first trying to estimate the difference $\Delta_t = x \oplus x_t$ for each t.

Roadmap to the Introduction: In Section 1.1 we detail our main results on using correlated instances in the CSP domain. We describe our techniques in Section 1.2. Section 1.3 contains additional results: showing how access to examples of correlated learning with parity (LPN) instances can speed up algorithms solving the underlying LPN, showing some limits on the benefit of correlated instances, showing the need for some "diversity" as well as correlation, and a proposal of a general complexity measure for algorithms which have access to auxiliary instances correlated via the solution space; Section 1.3 also offers some discussion and interpretation of our work. Section 1.4 contains a description of related work.

1.1 Main Results

Our main results are algorithms for solving CSP instances that come from widely-believed hard-to-solve distributions, given a number of correlated instances. Our results suggest that significant computational gains can be made when we can find multiple instances of search problems whose solutions are highly correlated *but not identical* – diversity pays off. The phenomenon of turning problems from intractable to efficiently solvable, goes way beyond exploiting coincidental algebraic structure to a wide setting of correlations in the realm of combinatorial problems.

[2]We also remark that in the above model each auxiliary instance for $t = 1, \ldots, T$ is generated independently of the others. Other models may allow different dependency structures, e.g. the t-th instance may "evolve" by perturbation of the $t-1$-st instance.

[3]In a cryptographic tampering attack, defined originally by Boneh, DeMillo and Lipton in 1997, an adversary is given $y = f(x)$ for a one-way function f; the secret x may be stored say on a smart card, and the adversary can induce a change on x without knowing it and then view $y' = f(x')$; different variants of the tampering model grants the adversary different degrees of control over the change induced. See Section 1.4 for more details on related work.

We have different results for solving correlated instances of CSPs when the primary instance is a worst case instance and when it is an average case instance. In both cases, we focus our attention to CSP instances that are fully satisfiable. [4] The average case distribution is based on a proposed one way function of Goldreich [Gol00]:

> **Random CSP distribution** rCSP**:** Select at random a right-d-regular bipartite graph $G = ([n], [m], E)$, $m = Dn$, and an input string $x \in \{0,1\}^n$, and compute y according to the mapping $f_{G,P}(x) = P \circ G(x)$ that computes each output bit by $y_k = P(x_{G(k,1)}, \ldots, x_{G(k,d)})$ for $k \in [m]$. Pictorially, we place the bits of x as labels to the input vertexes $[n]$, and then compute each output bit $k \in [m]$ of $P \circ G(x)$ by applying the predicate P to the substring obtained from the labels of its d neighbors.

This is a natural distribution on pairs of CSP instance and solution (where the solution satisfies all of the clauses) We point to an ongoing line of work studying average case CSPs as well as their approximate version, either conjecturing their hardness [Fei02, Ale03, ABW10, App12, BKS13], or studying various aspects of these problems [BQ12, App12, OW12, CEMT09, BQ12]. In particular, Goldreich [Gol00] conjectured that for some predicates P, it is hard to invert the image $f_{G,P}(x)$ with any non-negligible probability for a randomly chosen graph G and input x, and for $D = 1$. Subsequent works [BQ12, CEMT09] also considered this construction for larger values of D.

The generating process we consider, $Gen_{\mathsf{Pri},\mathsf{Aux}}$, samples (G, y) from a primary distribution Pri which will either be some arbitrary worst-case distribution over satisfiable instances, or the average-case distribution rCSP. The generating process then generates each auxiliary instance (G, y^t) by perturbing x to obtain x^t, and then setting $y_t = \mathsf{Aux}(x_t) = G \circ P(x_t)$.

Algorithms using *many* correlated instances

Our first result is that for a primary instance drawn at random, and for a large enough number of correlated instances $T = O(\log n)$, there is a randomized polynomial time algorithm that recovers a solution that satisfies $1 - \delta$ fraction of the constraints in an *average-case* primary instance, for arbitrarily small $\delta > 0$.

Informal Theorem 1 - random CSP *Fix any non-constant predicate $P : \{0,1\}^d \to \{0,1\}$ and constant $\delta > 0$. For small enough constant $\varepsilon = \varepsilon(d) > 0$ and large enough $D = D(d, \varepsilon, \delta)$ there is some $T = O(\log n)$ such that the following holds. There is a polynomial time algorithm that on input $(G, y), (G, y^1), \cdots, (G, y^t)$ finds a string $\tilde{x}$ that satisfies at least $1 - \delta$ fraction of the constraints in (G, y), with probability $1 - O(1/n)$. The instances $G, y, y^1, \cdots y^T$ are generated at random by the process $Gen_{\mathsf{Pri},\mathsf{Aux}}(n, T, \varepsilon)$ with $\mathsf{Pri} =$ rCSP and $\mathsf{Aux} = P \circ G$ (this notation means that Aux perturbs the solution x to get x^t and then applies $G \circ P$ on x^t).* [5]

Note that besides having sufficiently many correlated instances, the algorithm also requires the correlation between instances to be sufficiently high (that is, ε is sufficiently small), and the density D of the instances to be sufficiently large. The concrete ranges of parameters appear in Theorem 4.1.

We next show that knowing the differences $\{\Delta_t = x_t \oplus x\}_{t=1,\ldots,T}$ between the hidden solutions of the primary and auxiliary instances grants the algorithm surprising power. It allows the algorithm to find an exact solution to any *worst-case* primary instance (with high probability over the

[4] The advantage of working with CSP instances that are 100% satisfiable is that it makes it easier to reason about how perturbation on the optimal solution affects the constraints. Studying CSP instances that are only, say 99%, satisfiable is an interesting open question.

[5] In fact, the algorithm additionally finds such an $(1 - \delta)$-approximate solution for each auxiliary instance.

random coins for generating the auxiliary instances). Moreover, the algorithm works with *arbitrary* correlation parameter $0 < \varepsilon < 1/2$ and density $D \geq 1$, as long as the number of auxiliary instances T is sufficiently large, but still logarithmic. Needless to say, without auxiliary correlated instances this problem is NP-hard even to approximate, due to the PCP theorem [AS98, ALM$^+$98].

Informal Theorem 2 - worst case CSP, known differences *For every $\varepsilon \in (0, 1/2)$, there is a polynomial time algorithm, such that for every predicate P, density $D \geq 1$, graph G, $y = P \circ G(x)$, and constant $r > 0$, the algorithm on input $(G, y, y_1, \cdots, y_T, \Delta_1, \cdots, \Delta_T)$ finds a solution $\tilde{x}$ for (G, y) with probability $1 - O(n^{-r})$, provided that $T = O(\log n)$ is a sufficiently large multiple of $\log n$. All instances are generated by the process $Gen_{\mathsf{Pri},\mathsf{Aux}}(n, T, \varepsilon)$ with Pri is fixed to output (G, y) and $\mathsf{Aux} = P \circ G$ and where $\Delta_t = x_t \oplus x$ for each t.*

The formal statement appears in Theorem 4.4.

The algorithm in Theorem 2 can be easily extended to inverting arbitrary computation $y = f(x)$ in $\mathcal{NC}^0$ given a logarithmic number of correlated instances (i.e. images $y^!, \ldots, y^t$) and the differences between hidden solutions (or pre-images). The intuition behind this is that finding a pre-image of y with respect to f is equivalent to solving CSPs with a more general form where every constraint is with respect to a possibly different predicate. More generally, in Section 4.4.2, we show an algorithm that inverts any function f with low output locality $d = O(\log n)$ in the "worst case", if a sufficiently large number $T = O(\varepsilon^{-d} \log n)$ of auxiliary instances and the difference between their hidden solutions are available. The algorithm runs in $\text{poly}(2^d n)$ steps. From the perspective of tampering, this result states that every function with low output locality, in particular Goldreich's OWF, is easy to invert under a very weak model of tampering; we refer the reader to Section 1.4 for a comparison with cryptographic tampering results.

Interestingly, the limitation of $d \leq O(\log n)$ is optimal. Our next result shows that there is no algorithm that solves worst case instances with larger arity, even with access to correlated instances, unless $NP \subseteq BPP$.

Claim 1 - NP-hardness of solving worst case CSP with arity $d = \Omega(\log n)$, known differences *There is a predicate P with arity $O(\log n)$ such that unless $NP \subseteq BPP$, there is no probabilistic polynomial time algorithm for solving the P-CSP problem when given an additional correlated instance.*

The proof of the lemma is simple. Take an NP-hard CSP, and replace each Boolean variable by a cloud of $c \log n$ new variables. Replace each predicate by a new one that reads all $d \cdot c \log n$ variables encoding the original d variables, and that accepts if the majority decoding of each cloud would have satisfied the original predicate. One can see that any solution x_0 for the original CSP gives rise to a solution x to the new CSP, in which the value of all members of a cloud is the same. A perturbation x' of this solution would, with probability $1 - 1/\text{poly}(n)$ yield the exact same y, i.e. with high probability $P \circ G(x') = P \circ G(x)$. Thus, under our randomly-perturbed-solution correlation model, the correlated instance (G, y') will be identical to the primary instance (G, y) with probability $1/\text{poly}(n)$, in which case, it does not help to receive the correlated instance and the difference between the hidden solutions x' and x. This holds with high probability $1 - 1/\text{poly}(n)$, even when $O(\log n)$ auxiliary instances are available.

This lemma demonstrates an interesting threshold of $\log n$ in the arity of computations for which correlation is helpful.

Algorithms using *a single* auxiliary correlated instance

The previous two theorems demonstrate the power of having many correlated auxiliary instances. The next natural question that arises is "does correlation help when only a few auxiliary instances are available?" Our next two results investigate this case, and show that when the primary instance is drawn from the random CSP distribution, then the availability of even *one more* auxiliary instance, already gives a non-trivial computational advantage, albeit only for a *subclass of predicates*. These results extend previous work by Bogdanov and Qiao [BQ12] who showed that in the random CSP distribution certain classes of predicates can be solved or approximated (even with no auxiliary correlated instances, and when the density is sufficiently high). Our results show that the class of solvable predicates grows with the addition of an auxiliary correlated instance.

Again we separate two cases, depending on whether we know the difference $\Delta = x' \oplus x$ between the primary solution x and the perturbed solution x'. Our results are based on *derivatives* of the predicate P. The derivative in direction σ is defined by $P^\sigma(a) := P(a) + P(a+\sigma)$. We limit ourselves to CSPs with predicates that come from the class $\mathcal{P}$ of predicates $P : \{0,1\}^d \to \{0,1\}$ for which there is some direction σ, and index $i^* \in [d]$ for which P^σ is non-trivially correlated with its i^*-th input bit, that is,

$$\mathcal{P} = \{P : \{0,1\}^d \to \{0,1\} \mid \exists\ \sigma \in \{0,1\}^d,\ i^* \in [d],\ \gamma > 0, \text{ such that } \Pr[P^\sigma(a) = a_{i^*}] \geq 1/2 + \gamma\}$$

We show that for this class $\mathcal{P}$ of special predicates, a random CSP instance becomes easy to solve, given only a single auxiliary instance and the difference (between the two solutions). Interestingly, this algorithm requires the hidden solutions to be either sufficiently correlated, or sufficiently *anti-correlated*, depending on the predicate under consideration. More specifically, for a special predicate P satisfying the above condition w.r.t. direction σ and index i^*, if $\sigma_{i^*} = 0$, the perturbation flipping probability $p^* = \varepsilon$ must be sufficiently small, and otherwise the flipping probability $p^* = 1 - \varepsilon$ must be sufficiently large.

Informal Theorem 3 - random CSP, with a single auxiliary instance and known difference *Let $P \in \mathcal{P}$ be as above with parameters σ, i^*, and γ. For any $\varepsilon \in (0, 1/2)$, let the flipping probability p^* be set as*

$$p^* = \begin{cases} \varepsilon & \text{if } \sigma_{i^*} = 0 \\ 1-\varepsilon & \text{else } \sigma_{i^*} = 1 \end{cases}$$

Assume that ε is sufficiently small, and D is sufficiently large. For every constant $r \geq 1$, there is a polynomial time algorithm $\mathcal{A}$ that on input $(G, y, y^1\Delta)$ finds a solution $\tilde{x}$ for (G, y) with probability $1 - O(n^{-r})$. All variables are generated by the process $Gen_{\mathsf{Pri},\mathsf{Aux}}(n, 1)$ with $\mathsf{Pri} = \text{rCSP}$ and $\mathsf{Aux} = P \circ G$ with perturbation parameter p^* and $\Delta = x^1 \oplus x$.

The formal statement appears in Theorem 4.2.

Furthermore, we show that even when the difference is not given, correlation still helps to solve random CSP with the above special predicates of the case $\sigma_{i^*} = 0$. In this case, the algorithm simply requires the correlation between the hidden solutions to be sufficiently large.

Informal Theorem 4 - random CSP, with a single auxiliary instance *Let P be a special predicate as defined above with parameters σ, i^*, and γ,* **and** *$\sigma_{i^*} = 0$. Assume that ε is sufficiently small, and D is sufficiently large. For every constant $r \geq 1$, there is a polynomial time algorithm $\mathcal{A}'$ that on input (G, y, y^1) finds a solution $\tilde{x}$ for (G, y) with probability $1 - O(n^{-r})$. All variables are generated by the process $Gen_{\mathsf{Pri},\mathsf{Aux}}(n, 1)$ with $\mathsf{Pri} = \text{rCSP}$ and $\mathsf{Aux} = P \circ G$ and with perturbation parameter ε.*

The formal statement appears in Theorem 4.3.

Algorithms for well-known "hard-by-design" instances, using a single additional correlated instance

The gain from one more auxiliary correlated instance is not limited only to solving random CSP instances. We next show two algorithms for solving well-known families of CSP instances, with the aid of a single auxiliary correlated instance. Our first result solves a family of NP-hard 3SAT instances generated by Håstad's PCP reduction [Hås01]. These instances are the hardest known instances in terms of approximation, yet, they become easy in the presence of correlation.

Informal Theorem 5 - Håstad's 3SAT instances, with a single auxiliary instance *Let $L \in NP$ and let $\delta, \varepsilon > 0$. There is a polynomial time reduction H (due to Håstad) mapping instances of L to instances of* 3SAT *such that given a yes instance $\varphi \in$ 3SAT it is NP-hard to find an assignment x satisfying more than $\frac{7}{8} + \delta$ fraction of the clauses of φ. Nevertheless, given also an auxiliary correlated instance $\varphi' \in$ 3SAT there is a polynomial time algorithm that finds an assignment satisfying 99% of the clauses in φ with high probability over the choice of φ'. The correlated instance φ' is generated as follows*

1. *Let $x \in \{0,1\}^n$ be a satisfying assignment for φ. Flip each bit of x independently with probability ε, obtaining x'.*
2. *Let φ' be the* 3SAT *formula obtained by deleting from φ all of the clauses that are unsatisfied by the assignment x'.*

Our focus on 3SAT is for explicitness only. Similar results probably hold for other predicates. A formal statement of the above informal theorem appears in Theorem 5.1.

Our second result inverts randomized encodings of [AIK04]. The notion of randomized encodings allows to "encode" a complex deterministic function f by a simple randomized function $\hat{f}$. From the output of the randomized encoding $\hat{f}(x;r)$ one can reconstruct the output of $f(x)$, and at the same time no other information of x is revealed; the latter is called the privacy property. Applebaum, Ishai, and Kushilevitz [AIK04] showed in a celebrated work how to represent any $\mathcal{NC}^1$ computable function by an $\mathcal{NC}^0$ function with output locality 4, denoted by $\mathcal{NC}^0_4$. This encoding is referred to as the AIK randomized encoding henceforth. By the privacy property, images $\hat{f}(x;r)$ of AIK encodings are hard to invert on the average over the random choices of r, as long as the original function image $f(x)$ is hard to invert.

AIK thus constructs cryptographic primitives such as stream-ciphers and commitment-schemes which have low depth and constant output locality yet are still hard to invert. (See the survey by Applebaum [App11] for more details.)

If we view both x and r as the input variables to the encoded function $\hat{f}$, then inverting $y = \hat{f}(x;r)$ corresponds exactly to solving a CSP instance, where every output bit is a constraint over 4 input bits that belong to x or r. The randomly-perturbed-solution model for CSP naturally extends to this setting, where the generating process $Gen_{\mathsf{Pri},\mathsf{Aux}}$ samples a primary instance $(\hat{f}, y = \hat{f}(x;r))$ together with many auxiliary instances $(\hat{f}, y^t = \hat{f}(x^t;r^t))$ with (x^t, r^t) perturbed from (x,r) by flipping each bit independently with probability ε. The algorithmic goal is still to invert the image $f(x)$ of the original function.

We show that if the original function f is "well formed" (see Section 6.4.1 for a precise definition, we mention only that any function can easily be modified to possess this property) then, its encoding is easy to invert in the worst-case when given a single auxiliary instance $y^1 = \hat{f}(x^1;r^1)$

Informal Theorem 6 - AIK randomized encodings, with a single auxiliary instance *For every function f in $\mathcal{NC}^1$, let $\hat{f}$ be the AIK randomized encoding of f. Fix any $\varepsilon \in (0, 1/2]$ and any*

constant $\alpha > 0$. There is a polynomial time algorithm, such that, for every well-formed *function f, string x, and sufficiently long r, the algorithm on input $(\hat{f}, y, y^1)$ inverts $f(x)$ with probability $1 - n^{-\alpha}$, where all variables are generated by the process $Gen_{\mathsf{Pri},\mathsf{Aux}}(n, 1)$ with* Pri *fixed to output $(\hat{f}, y = \hat{f}(x; r))$ and* Aux $= \hat{f}$.

See Theorem 6.3 for a formal statement.

We compare this result with Theorem 3 and 4 for solving random CSPs with a single auxiliary instance. There, the predicate under consideration comes from a special class, but the CSP graph is chosen at random with the only constraint that it must have high density. Here, given the original function f, the AIK encoding scheme determines the set of predicates and the graph based on f; in particular, the graph is not random and does not have high density. Therefore, Theorem 5 can be view as another demonstration that even a single correlated instance helps solving special classes of CSPs.

To summarize, these results demonstrate that access to several correlated instances gives significant more algorithmic power than considering one instance alone, and it can turn a conjectured intractable problem into a provably tractable one.

1.2 Techniques

We next describe the high-level techniques that we have used for the results outlined above.

The shift encoding. We start with the easiest algorithmic setting, where to solve a primary instance G, y, the algorithm is given multiple auxiliary instances $(G, y^1, \cdots, y^T)$ *and* the differences $(\Delta^1, \cdots, \Delta^T)$ where $\Delta^t = x \oplus x^t$ is the difference between the primary solution x and the perturbed solution x^t. In this setting, we are able to recover x completely even if the primary instance is a worst case instance, and for any correlation $\varepsilon \in (0, 1/2]$ and density $D \geq 1$.

The key observation is that given sufficiently many correlated instances, a fixed output vertex $k \in [m]$ will eventually see all perturbation patterns: Its value in the t^{th} instance is equal to

$$y_k^t = P(x_{G(k)}^t) = P(x_{G(k)} \oplus \Delta_{G(k)}^t)$$

where we denote by $x_{G(k)}$ the restriction of x to the d indexes neighboring k. For every instance t, the difference $\Delta_{G(k)}^t$ (restricted to the neighbors of k) is a d-bit string, where every bit $\Delta_{G(k,l)}^t$ is independently Bernoulli distributed with probability ε of being 1. Given sufficiently many auxiliary instances, every possible "shift" $a \in \{0,1\}^d$ will appear (i.e., for every $a \in \{0,1\}^d$, $\exists t$ s.t. $\Delta_{G(k)}^t = a$). Since the differences $\{\Delta_{G(k)}^t\}$ are known, the algorithm can collect the values of P on every possible shift a from $x_{G(k)}$, which form an encoding of $x_{G(k)}$; we call it the *shift-P-encoding* of $x_{G(k)}$. More specifically, let

$$\text{Encode}(z) \triangleq (P(z \oplus a))_{a \in \{0,1\}^d} \triangleq \left[P(z \oplus \underbrace{0 \cdots 0}_{\text{d}}), P(z \oplus \underbrace{0 \cdots 0}_{\text{d-1}} 1), \cdots, P(z \oplus \underbrace{1 \cdots 1}_{\text{d}}) \right] \in \{0,1\}^{2^d}$$

Given the shift-P-encoding $\text{Encode}(x_{G(k)})$, if the predicate P satisfies that the encoding for every $z \in \{0,1\}^d$ is unique, then we can uniquely recover $x_{G(k)}$. (This can be done efficiently, since the encoding has constant size 2^d.) Then using the same procedure for every output bit k, the algorithm uncovers the unique value of $x_{G(k)}$ for every k. By stitching them together, it obtains the unique string x consistent with y as well as all $\{\Delta^t\}_t$ and $\{y^t\}_t$.

Does this approach handle all predicates? No, there are predicates P for which the shift P encoding is not unique. The next challenge is *how to go beyond predicates that admit unique shift*

encoding and handle all non-constant predicates. The difficulty lies in that for a general predicate P, there may exist different strings $z \neq z'$ that have the same encoding $\mathrm{Encode}(z) = \mathrm{Encode}(z')$. Then, from the encoding $\mathrm{Encode}(x_{G(k)})$, the algorithm can only deduce that there are multiple candidates for the value of $x_{G(k)}$. However, given a collection of candidates for all sub-strings $\{x_{G(k)}\}$, it is not clear how to find a single string $\tilde{x}$ that simultaneously satisfies all constraints on all sub-strings $\{x_{G(k)}\}$.

We overcome this problem by showing that given a valid encoding E (for which there exists a pre-image), all the pre-images of E form a unique affine space Λ. Then, given encoding $\mathrm{Encode}(x_{G(k)})$, the constraints on $x_{G(k)}$—that is, $\{P(x_{G(k)} \oplus \Delta^t_{G(k)}) = y^t_k\}_t$—are equivalent to a set of linear constraints on $x_{G(k)}$, that is, $x_{G(k)} \in \Lambda$. Now, given the collection of linear constraints for all k, the algorithm can solve the linear system efficiently to recover a string $\tilde{x}$ consistent with all constraints.

These are the main ingredients in the proof of Theorem 2.

Estimating the differences, and then estimating the estimation error. For Theorem 1, we adapt the algorithm above to the case where the differences Δ^t are not given. Here the natural approach is to *estimate* the differences, and obtain $\tilde{\Delta}^1, \ldots, \tilde{\Delta}^T$ such that $\tilde{\Delta}^t \approx \Delta^t$ for $t = 1, \ldots, T$. The estimation is possible because by knowing which output bits have flipped we get a reasonable guess as to which input bits have flipped. The smaller the arity d, and the higher the density D, the better this estimate is.

However, replacing Δ^t by $\tilde{\Delta}^t$ is reasonable only if our algorithm above is resilient to the noise introduced by the approximation. This does not hold, unfortunately, because the second step of the algorithm relies on solving linear equations, a procedure that is infamously non-robust to noise. Nevertheless, we overcome this problem by taking advantage of the fact that our primary instance comes from the random CSP distribution. We add an estimate-verification step that marks the indices $i \in [n]$ that suffered from unbounded estimation errors (informally, this refers to the case where errors in the i-th bit occur for Δ^t across too many t's). Then, we prove that this verification procedure will, with very high probability, not make mistakes, assuming that the primary instance is drawn from the random CSP distribution. Finally, by avoiding these problematic indices we can correctly recover most of the solution.

Directional derivatives. In case only a single auxiliary correlated instance is available, we cannot expect to learn the shift-encoding of any substring of x. Instead, we observe that by getting an evaluation, for each $k \in [m]$, of $P(x_{G(k)})$ and $P(x'_{G(k)}) = P(x_{G(k)} + \Delta_{G(k)})$, we can directly compute $P^\sigma(x_{G(k)})$ for $\sigma = \Delta_{G(k)}$. The extra information provided by P^σ allows us, in Theorems 3 and 4, to solve a broader class of CSPs. We rely on the randomness of the primary instance to ensure that our difference estimation is correct, without which the derivatives will be useless.

Solving NP-hard instances using the structure of the solution space. In Theorem 5 we solve instances that come from PCP reductions, with the help of an extra correlated instance. Here the given primary instance is not random and so difference estimation as in the previous results will not necessarily work. Our approach is to rely on the fact that the satisfying assignment, if it exists, is highly structured. In particular, the variables are partitioned into constant-size blocks (corresponding to long code encodings) and on each block there is only a small number of possible assignments. It is feasible to enumerate all possible legal assignments on a pair of blocks and to check if they are likely to give rise to a given difference pattern. We prove that this works by analyzing the precise long code test and showing that certain difference patterns are more likely for certain assignments than for others.

Inverting AIK randomized encoding by cascading from many break points. In Theorem 5, we invert the AIK randomized encoding given a single auxiliary instance. The first step of the inverting algorithm is similar to the algorithms for solving CSP: By crucially relying on the low output-locality feature of AIK, it recovers some "local" information about the hidden solution. (In the case of CSP, the local information pieces pertain to the substrings $\{x_{G(k)}\}$; similarly, in AIK, these local information pieces are about subsets of bits in x and/or r influencing different output bits.) However, unlike the case of CSP, these local information pieces are not sufficient for generating the global solution. The graph corresponding to the AIK encoding is not random and its density is very low, so this first step can only recover an ε fraction of the solution. In remedy, the algorithm in a second phase, relying on the concrete structure of the AIK encoding, uses the partial information as *break points* for starting many iterative *cascading steps*. In each step given the appropriate information associated with one input bit, the value and associated information of the input bit "next to" it are found; thus, the information recovered propagates, until the whole input is found.

The concrete procedure for recovering the initial local information and cascading information requires heavy engineering on the specifics of the AIK encoding. Nevertheless, the overall structure of the inverting algorithm is the key and demonstrates that the ways to leverage correlation can vary from problem to problem.

1.3 Additional Results and Discussion

A Complexity Measure for Correlated-Instances. One of the goals of research in the design and analysis of algorithms is to develop algorithms which work well in practice, by taking into account all available data. In the CSP domain, we demonstrated that access to multiple problem instances with correlated solutions can change the complexity of problems from intractable to tractable. It is similarly possible that access to correlated instances can lead to faster and different algorithms for problems which are already solvable in polynomial time.

To this end, we propose a new *correlated-instance* complexity measure. Correlated-instance complexity measures the performance of an algorithm on a primary instance given as auxiliary input a tuple of correlated instances where the correlation is over the solution space. We emphasize that the measure does not restrict the distribution of the primary instance, which can be worst-case or average-case, but introduces the new dimension of correlation, and can be used to analyze the behavior of algorithms that utilizes multiple correlated instances. See Section 2.

Correlation versus Diversity We have demonstrated that correlation between instances is helpful. Yet, interestingly, "too much correlation" is not good either. Clearly, if the auxiliary instance at hand is identical, thus 100% correlated, to the primary instance, it is of no use, as it brings no additional information. Hence, some amount of diversity is necessary.

We further observe that in the randomly-perturbed solution model of CSP, the amount of "diversity", i.e. the perturbation parameter ε, should be some positive constant for the auxiliary instances to be useful. Indeed, we observe that this follows from the *exponential-time-hypothesis* [IP01] which conjectures that for every $k > 2$ the best algorithm for k-SAT requires time $2^{s_k n}$ for some constant s_k.

Claim 2 - hardness of solving correlated kSAT, when correlation is too high *Assuming the exponential time hypothesis, there is no polynomial-time algorithm for solving a primary kSAT instance φ even if given a randomly perturbed*[6] *correlated instance φ' obtained by perturbing the*

[6] We remark that the distribution of φ' conditioned on φ is not exactly the same as in our model, but this difference

solution of φ with perturbation parameter $\varepsilon = o(1)$ and then modifying the affected clauses to make φ' satisfiable.

The reason is that an algorithm can generate the correlated instance (G, y') on its own in time that is $\exp(\tilde{o}(n))$ (simply by cycling through all possibilities of having $o(n)$ affected clauses).

Going back to our techniques, intuitively we are using the differences between the two instances (i.e diversity) to our advantage, to get a handle on the derivative *in addition* to the original function, which allows us to approximate it better. Too large a correlation would not yield enough information about the derivative, whereas too little correlation is hard to tell apart from noise. We find the non-monotonic behavior of the problem very interesting and worthy of more careful examination.

Does Correlation Always Help? We have demonstrated that significant computational gains can be made for solving CSPs, given access to correlated instances. One basic question that arises immediately is "do *all* search problem become easy, given $I, I_1, \cdots, I_T$ for sufficiently and naturally correlated x_i's?"

Restricting the question to the context of CSPs, although we described a number of algorithms for solving CSPs, these do not cover all possible cases. In particular, they do not address the case of CSPs whose optimal solution does not satisfy all constraints. Here, for the 3LIN predicate, we have the following hardness result.

Claim 3 - hardness of solving correlated 3LIN instances that are $1-\delta$ satisfiable *Unless $P = NP$, there is no polynomial-time algorithm that gets as input a 3LIN instance (G, y) whose optimal solution satisfies at least $1-\delta$ of the clauses, and a correlated instance (G, y') generated using $\mathsf{Aux} = P \circ G$ (where $P = 3LIN$), and finds a solution that satisfies more than $1/2 + \delta$ of the clauses.*

Proof. We know [Hås01] that it is NP-hard to decide if a given instance (G, y) has a solution satisfying at least $1-\delta$ of the clauses, or no more than $1/2+\delta$. Given an instance G, y, we can use A to solve it, by simulating the instance y'. First we generate a random difference vector $\Delta \in \{0,1\}^n$ by setting each bit independently to 1 with probability ε, and then we let $y' = y + P \circ G(\Delta)$. The answer of A on (G, y, y') is a solution to the original instance. It remains to observe that for $x' := x + \Delta$ we have $y' = P \circ G(x')$ so (G, y') is distributed exactly according to $\mathsf{Aux} = P \circ G$. □

One wonders whether a similar hardness result exists for a CSP with perfect completeness.

This result shows that there are some CSP-based problems for which there is no algorithmic gain from looking at additional correlated instances. In contrast, we remark that for every function f, there is another "equivalent" function f' that is susceptible to correlated instances. More accurately, f' is as hard to invert as f, and yet becomes easy to invert when given an additional auxiliary instance. The reduction is as follows[7]. Given any function $f(x)$, we define the corresponding randomized function $f'(x; r, s)$ as follows: f' on input x and $2kn$ bits (k is set later) of random coins $r = \{r_{ij}\}_{i\in[n],j\in[k]}$ and $s = \{s_{ij}\}_{i\in[n],j\in[k]}$, outputs $f(x)$ and an "encoding" $\{y_{ij}, r_{ij}\}_{i\in[n],j\in[k]}$ of the input x using the random coins r and s, where every input bit x_i is encoded as $\{(y_{ij}, r_{ij})\}_{j\in[k]}$ with $y_{ij} = x_i \cdot r_{i,j} + s_{i,j}$. It is easy to see that given uniform random coins r_{ij}'s and s_{ij}'s, the encoding hides the input x information theoretically; thus, the randomized function $f'(x; r, s)$ has the same inversion complexity as $f(x)$ (when r, s are randomly chosen).

is not important. In particular, it is reasonable to expect the ETH to hold for CSPs (G, y) as in our definition, for some predicate P, in which case an equivalent formulation could be made in which the correlated instances are exactly as in our model.

[7]We thank an anonymous referee for pointing out a simpler proof of this claim.

We show that $f'(x;r,s)$ becomes easy to invert given a single correlated instance $f'(x';r',s')$ with ε-perturbed x', r', s', for a constant ϵ. The inversion procedure simply tries to recover the x from the two input encodings $\{y_{ij}, r_{ij}\}$ and $\{y'_{ij}, r'_{ij}\}$ contained in the two instances as follows: Collect the j's for which $r_{i,j} = 1$ but $r'_{i,j} = 0$. For these j's, the value $y_{i,j} \oplus y'_{i,j} = x_i + s_{i,j} + s'_{i,j}$ is correlated with x_i, and can be viewed as vote for the value of x_i. Thus by taking a majority vote, x_i can be recovered with high probability $1 - O(1/n)$, provided that k is a sufficiently large logarithmic number. Therefore, we are likely to recover all the bits of x.

We remark that key of the above transformation from f to f' is the encoding of the input, which is information theoretically hiding given one encoding, but easy to decode given two. It is easy to see that this encoding can also be used to transform any search problem to another search problem with the same complexity (under an input distribution where the random coins used for the encodings are indeed random), which becomes easy to solve given a correlated instance.

The Benefit of Correlation in Learning Theory Access to correlated instances is a wider phenomenon beyond improving time complexity of algorithms. In particular, it seems that in subareas as varied as *learning theory*, *coding theory* and *biological experiments*, correlated problem instances come up naturally. Does the benefit of correlation extend as well? We suggest an affirmative answer showing an example in learning theory.

The classical PAC learning [Val84], considers an underlying hidden concept C (for example, C can be a DNF, a junta, or a parity function), a learning algorithm is given multiple labelled examples of the form $(x, C(x))$, and the goal is finding a hypothesis that labels future examples correctly with high probability (w.r.t. a distribution on examples). In the commonly used random-query model, the learning algorithm sees only a sequence of examples $(x_1, C(x_1))$,$(x_2, C(x_2)), \ldots$ where each x_i is independently drawn from a probability distribution.

Two degrees of correlation may be incorporated into the random query model. First, the examples may be correlated. Formally, a generating process Gen_C produces a sequence of pairs (I_i, S_i) where $S_i = C$ for all i, and $I_i = (x_i, C(x_i))$ for correlated x_i's. This model has been studied in the learning literature (e.g.[BMOS05]). However, not only the examples may be correlated, in actuality, many concepts to be learned are potentially correlated themselves, and leveraging this correlation may reduce the complexity of learning any one of them in isolation. Think of multiple specimens (the concepts to be learned) which are highly correlated but not identical, all exposed to a battery of experimental conditions, as a researcher/learner is trying to learn hidden variables governing their reaction. Put in our framework, there is a generating process Gen that generates correlated concepts C_0,C_1,...,C_T, and then provides a list of examples $I_t = \{x_i, C_t(x_i)\}_{i=1}^k$ for each $t = 1, \ldots, T$, with the corresponding solutions $S_t = C_t$. This is best imagined as a table whose rows are indexed by the examples and whose columns are indexed by the various correlated concepts. The learner gets to see the entries of this table, describing the behavior of each concept $C_1, \ldots, C_T$ on the same set of examples $\{x_i\}$ (corresponding to the experimental conditions), and tries to find a hypothesis for each of the hidden concepts.

We illustrate the strength of this model for the learning parity with noise (LPN) problem. In the standard LPN, there is a hidden function $C(x) = \langle s, x\rangle \mod 2 = \sum_{i=1}^n s_i x_i \mod 2$, and the learner gets noisy samples of the form $x, C(x) + b$ for independently drawn $x \in \{0,1\}^n$ and a random biased bit b that equals 1 with probability $\frac{1}{2} - \delta$. The goal is to find $s \in \{0,1\}^n$. In our correlated concept model, there is a random string s_0 and T ε-noisy copies of it, $s_1 \ldots, s_T$; these strings describe $T + 1$ parity functions $C_t(x) = \langle s_t, x\rangle \mod 2$. The learning algorithm gets as input a table Y whose rows are indexed by the queries $x_1, \ldots, x_k$ and whose columns are indexed

by $s_0, \ldots, s_T$ such that $Y_{it} = \langle x_i, s_t \rangle + b_{it}$, where b_{it} are independent biased noise bits[8], and the goal is to find the strings $s_0, \ldots, s_T$.

We show that when the underlying parity functions are sufficiently correlated, with $\varepsilon = 1/n$, the LPN problem becomes easy to solve. The same statement can also be shown for larger ε using a similar approach, with the running time growing exponentially in εn. For simplicity, we only analyze the case with $\varepsilon = 1/n$.

Claim 4 - learning correlated parities *There is a polynomial-time algorithm that, given access to $k = O(\log n)$ samples from $T = O(n \log n)$ parities with $1/n$-noise, learns s_0 with high probability.*

Let us sketch a proof of this claim. Let Y be the input arranged in a table, so that $Y_{it} = \langle x_i, s_t \rangle + b_{it}$. Our first step will be to find the difference $s_0 + s_t$ for many t's, by a guess-and-check strategy. Note that by our choose of $\varepsilon = 1/n$, it is likely that $s_0 + s_t$ has very low weight. Let $v_i = \langle x_i, s_0 \rangle + b_{i0} + \langle x_i, s_t \rangle + b_{it} = \langle x_i, s_0 + s_t \rangle + (b_{i0} + b_{it})$. Let $\Delta_{0,t}$ be a guess for $s_0 + s_t$. We will check whether our guess is correct by computing v' defined by $v'_i = v_i - \langle x_i, \Delta_{0,t} \rangle$. This vector will be biased towards 0 only if the guess is correct (since in that case the i-th entry equals $b_{i0} + b_{it}$ which is 1 with probability $\frac{1}{2} - 2\delta^2$). Searching among all difference vectors of weight 1 we will succeed with constant probability and thus be able to find the difference for many "good" columns t. For simplicity let us pretend all differences had weight 1.

Our second step will be to find s_0. Fix i, and observe that for each "good" t we know $s_0 + s_t$ so can check if $\langle s_0 + s_t, x_i \rangle = 0$. For each such t, the entry Y_{it} in the input is an independent noisy copy of $\langle s_0, x_i \rangle$. Taking majority, we can get a guess for $\langle s_0, x_i \rangle$ that is correct with probability $1 - 1/\mathrm{poly}(n)$. Collecting such equations for various x_i we can solve a linear system to recover s_0. This also immediately gives s_t for all good t's.

A Possible Interpretation: Why do we Collaborate? Finally, we mention that the computational benefit of access to instances which are correlated via the solution space may give computational insight on a fundamental "human" question: why do we interact and collaborate with each other?

Collaboration between different entities which have access to different information has been an over riding theme in theoretical computer science research in the last few decades. Famous examples include *communication complexity* and *multi party secure computation*. In both of these examples, interaction and collaboration is built-in the definition of the problem, as the goal is to compute a function of the information held by "the other party". However, as individuals and scientists it seems that we benefit from collaboration not only as a means to find out functions of what our partners know but in order to be able to understand ourselves better and achieve more than we could on our own. A fascinating question is: why and when is collaboration beneficial from a computational point of view?

One interpretation of our results (and actually an initial motivator for our work) is that *correlation* between the inputs of the different parties may be by itself a major driving force for interaction and collaboration. We may view collaborating parties A and B as holding respectively inputs $y = f(x)$ and $y' = f(x')$. Each party knows her own y (and y' respectively) but doesn't know the hidden variables (or *internal state*) x (and x' respectively). They collaborate by exchanging

[8]Note that if b_{it} are identical for all t rather than being independent then this model reduces back to the standard LPN problem. The reason is that upon receiving $x, C_1(x)$ the learner can always generate $C_1(x) + \langle x, \Delta s \rangle + b_1$ for any string Δs of his choice. This clearly equals $C_2(x) + b_2$ for $\Delta s = s_2 - s_1$.

$f(x)$ and $f'(x')$ to engage in "self discovery". If their internal states are sufficiently similar (correlated) but not identical, collaboration will result in successful computation of x and x'. Similarly, collaboration with multiple entities can be of further benefit.

We do note that in our setting, once A and B exchange $y = f(x)$ and $y' = f(x')$, they do not need to interact further. It would be highly interesting to find examples where more complex interaction would be useful for discovering the hidden internal state. In other words, settings in which the algorithms of A and B would benefit from the adaptivity of the interaction. For example, a variant of the above would be a setting in which the access to y is limited compared to the length of x and y. Say x, y are exponentially long in the running time allowed to the recovery algorithm. Then, the recovery algorithm may be given "oracle" access to the tuple (f, y, y'_i). Given a set of indices $J = \{j\}$, the goal would be to recover the j-th bit of x for all $j \in J$. We leave as an open question to exhibit functions for which an adaptive inverting algorithm can be more powerful than non-adaptive inverting algorithms, necessitating a multi-round interaction. The answer is far from obvious.

1.4 Other Related Work

On Correlation in Cryptanalysis Throughout our work we assumed that the CSP-solving algorithms *magically* have access to a generating process which produces instances with correlated solutions. As alluded to earlier, in some settings, one may imagine being able to actively obtain such instances. One such setting studied in the cryptographic literature is called *tampering* attacks. The first tampering attack was introduced in a beautiful paper by Boneh, DeMillo and Lipton in 1997 "On the Importance of Checking Cryptographic Protocols for Faults" who asked whether one may induce hardware faults which would result in viewing the results of running a cryptographic algorithm with a "modified" (or tampered) secret key and thus help to recover the original unmodified secret key. More recently, the works of [DOPS04, ACM+13] consider adaptive tampering attacks on the randomness used by a cryptographic algorithm and show that achieving security is impossible against such strongly adaptive attacks. We note that the correlation models we considered for CSPs are by and large, much weaker than the tampering considered in the cryptographic literature.

Another body of cryptanalytic work in which correlated instances play a crucial role is differential cryptanalysis [BS90, BS91c, BS91b, BS91a, BS92, BAB93] which has been extensively applied to symmetric key encryption schemes such as AES, with the goal of recovering the secret key. The analysis works on the premise that the cryptanalyst has access to many correlated $(plaintext, ciphertext)$ pairs. By analogy in our work the adversary sees instances $(G, y), (G', y')$ (which can be thought of as the "ciphertexts") and does *not* see the correlated solutions x, x' (which can be thought of as the "plaintexts").

Yet another related body of work is on related-key attacks in the cryptography literature, where the adversary may even *choose* the correlation between the secret key or inputs [Bih94, Knu92, BK03, GL10, BC10, BCM11, AHI11, Wee12].

On the flip side, Rosen and Segev [RS10] show that families of lossy trapdoor functions, introduced by Peikert and Waters [PW11], are actually secure under natural correlated products. In the same vein are the works on *aux-input security* [DKL09, DGK+10]. The goal is to design cryptographic primitives with secret keys, which are secure even if the adversary has access to auxiliary information on the secret key in the form of hard to invert computations computed over (e.g. one way functions of) the secret key. In a sense, our results show that Goldreich one-way functions and the AIK transformations are not auxiliary-input secure.

Cryptanalysis on Goldreich's OWF As mentioned earlier the work of Bogdanov and Qiao [BQ12] on the (in)security of Goldreich's OWFs is particularly relevant, and is a technical starting point for our work. They show that Goldreich's OWFs are easy to invert *in the standard model* (where algorithm is given only one instance) in two restricted settings: The first is for arbitrary predicates but assumes that the algorithm is given a 'hint' which is a string x' that is correlated to x. The second is an inversion algorithm that works for predicates correlated with one or two of their input bits. In comparison, our first algorithm (for Theorem 1) addresses arbitrary predicates given hints of a much weaker nature, that is $f(x')$ rather than x'. Our third algorithm (from Theorem 3) which takes a single $f(x')$ hint is limited to predicates in $\mathcal{P}$. For the predicates in this class, not covered in the first result of [BQ12], one additional correlated instance already makes a difference.

The Study of Correlated Examples in Learning Theory In section 1.3 we showed how receiving examples of correlated concepts to be learned can be beneficial. Access to correlated examples of the *same concept* have been previously studied. For example Bshouty et. al. [BMOS05] show an efficient learning algorithm for DNFs in the, so called, random walk query model, where the $i+1^{\text{th}}$ query x_{i+1} is derived from x_i by performing a random bit flip. In contrast, learning DNFs in the random query model is a notoriously hard problem. Thus, the correlation between the queries x_i's seems to give the learner significantly more power.

The Study of Correlated Examples in the Planted Independent Set Model In work inspired by this paper, Holmogren (private communication) recently examines correlated instances in the context of the planted independent-set semi-random graph model of Feige and Kilian [FK01].

The semi-random graph model [FK01] was initially proposed to try to "mediate between the unstructured "uninteresting" graphs produced by the purely random models and the worst-case graphs that are seemingly beyond the heuristics's ability to solve." To this end, the model first plants in their graphs a solution to an underlying graph problem (e.g. independent set, coloring, graph bisection) as in the random graph world. Next, *subject to this planted solution remaining intact*, the model adds edges to the graph in a random fashion followed by a final adversarial step in which edges can be added arbitrarily. In the case of planted independent set of size αn for an n node graph, [FK01] proves a sharp threshold between when it is easy to recover the planted independent set and when it is NP-hard, depending on the probability of adding edges in the random step.

A natural question to ask in the context of studying correlations defined over solutions, is whether access to *two* graphs with correlated planted independent sets (i.e correlated solutions), change the complexity of the problem when it is NP-hard. Indeed, Holmogren presents an algorithm that given a single auxiliary graph, finds an $(1-\gamma)$-approximate independent set (of size at least $(1-\gamma)\alpha n$) in the primary graph, for any constant γ. There, the auxiliary graph is correlated with the primary graph in the way that its planted independent set overlaps with that of the primary graph for more than $(1-\delta)\alpha n$ vertexes, for a sufficiently small $\delta = \delta(\gamma)$ (and the rest of the graphs are constructed independently according to the semi-random graph model above).

Correlated Instances in Smooth Analysis The celebrated work on smooth analysis by Spielman and Teng[ST04] introduced a new measure on the complexity of algorithms, first illustrated via the smooth analysis of the celebrated Simplex algorithm in the realm of real valued inputs. Smooth analysis measure for an input x , is the expected behavior of the algorithm on correlated inputs which are the result of subjecting x to perturbations (e.g. flip its bits with a certain probability). The correlated instance complexity measure we introduce, is different in several aspects. First, we don't consider inputs which correlated to a primary input x, but rather inputs whose underling

solutions correlated to the solution of x. Second, we deviate from the traditional paradigm of an algorithm working on a single input x, toward the design of algorithms which receive input x as well as in addition auxiliary inputs x_i's whose solutions are correlated to the solution to x, to assist in the goal of computing the solution for x.

We mention that Spielman and Teng [ST03] in a follow up work to their original smooth analysis paper observe that in the discrete input domain, perturbations of the input "should probably be restricted to those which preserve the most significant aspect input with respect to a given situation". To address this, they define property-preserving perturbations to inputs and relate this measure to property testing work [GGR96] and the heuristics of Feige and Kilian [FK01] for finding cliques on semi-random graphs with planted cliques.

1.5 Organization

In the rest of the paper, we first introduce some notions and definitions in Section 2. In Section 3, we propose a complexity measure for solving correlated instances of general problems. In Section 4, we design algorithms for solving (mostly) random CSPs; for ease of exposition, we first describe the algorithms that are given the differences between hidden inputs (proving informal Theorem 2 and 3), and then describe algorithms that do not rely on knowing the differences (proving informal Theorem 1 and 4). In Section 5, we show how to solve Håstad's 3SAT instances given a single correlated instance. Finally, in Section 6, we show how to invert the randomized encodings of well-formed functions given correlated instances.

1.6 Acknowledgement

We would like to thank the anonymous reviewers for many valuable suggestions and comments.

Contents

2 Preliminaries

Let X and Y be two random variables over $\{0,1\}$. We say that X is ε-balanced if $|\Pr[X = 0] - 1/2| \leq \varepsilon$. The *correlation* between X and Y is the following value

$$\Pr[X = Y] - \Pr[X \neq Y] = 2\Pr[X = Y] - 1$$

A bit string $x \in \{0,1\}^n$ is ε-balanced if its i^{th} bit is ε-balanced with i chosen at random from $[n]$. The correlation between two bit strings x and y in $\{0,1\}^n$ is defined as the correlation between the i^{th} bit of x and y, with i chosen at random from $[n]$. The Hamming weight $\mathsf{Wt}(x)$ of a bit string x is the number of 1's in x, and relative Hamming weight $\mathsf{wt}(x)$ of x is the fraction of 1's in x, $\mathsf{wt}(x) = \mathsf{Wt}(x)/|x|$. The Hamming distance $\mathsf{Dis}(x, x')$ between two strings x and x' of equal length is the hamming weight of their difference $\mathsf{Dis}(x, x') = \mathsf{Wt}(x \oplus x')$, and the relative Hamming distance $\mathsf{dis}(x, x')$ is the Hamming distance normalized by the length of the strings $\mathsf{dis}(x, x') = \mathsf{Dis}(x, x')/|x|$. If $\mathsf{dis}(x, x') \leq \tau$, we say that x' is τ-close to x and vice versa. An ε-balanced string x satisfies $1/2 - \varepsilon \leq \mathsf{wt}(x) \leq 1/2 + \varepsilon$. The correlation between x and x' is exactly the relative hamming distance between them.

Function Notation. Let P be a predicate from $\{0,1\}^d \to \{0,1\}$. The correlation between P and its i^{th} input bit for $i \in [d]$ is the correlation between the random variables $P(x_1, \cdots, x_d)$ and x_i, where $x_1, \cdots, x_d$ are sampled randomly and independently from $\{0,1\}$. We say that P is ε-balanced if the random variable $P(x_1, \cdots, x_d)$ is ε-balanced, and simply balanced when $\varepsilon = 0$. The influence of a set $I \subseteq [d]$ of input bits on the predicate, denoted as $\mathrm{Inf}_P(I)$ is the value

$$\mathrm{Inf}_P(I) = \Pr[x \xleftarrow{\$} \{0,1\}^d \ : \ P(x) \oplus P(x^{\oplus I})]$$

where $x^{\oplus I}$ is the string that differs from x at I. For simplicity of notation, we denote by $\mathrm{Inf}_P(i)$ the influence of the i^{th} input bit on P.

The derivative of P with respect to the i^{th} input bit x_i, denoted as P^i, is the function

$$P^i(x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_d) = P(x_1, \cdots, x_{i-1}, 0, x_{i+1}, \cdots, x_d) \oplus P(x_1, \cdots, x_{i-1}, 1, x_{i+1}, \cdots, x_d)$$

The boundary β of P is defined as follows:

$$\beta = \Pr_{z \xleftarrow{\$} \{0,1\}^d} [\exists z', \ \mathsf{Wt}(z - z') = 1 : P(z) \neq P(z')]$$

A function $\mu : \mathbb{N} \to \mathbb{R}$ is said to be *negligible* if $\mu(n) < n^{-c}$ for any constant $c > 0$ and sufficiently large n. We say that a function $f : \{0,1\}^* \to \{0,1\}^*$ or a circuit C is d-local if every output bit depends on at most d input variables, and d is called the output locality of f or C. For any function $f : \{0,1\}^* \to \{0,1\}^*$, we denote by f^n the function restricted to inputs of length n.

Probability Notation. Let U_n denote a random variable that is uniformly distributed over $\{0,1\}^n$. If X is a probability distribution, we write $x \xleftarrow{\$} X$ to indicate that x is a sample taken from X, and if S is a set, we write $x \xleftarrow{\$} S$ to indicate that x is uniformly selected from S. Similarly, if f be a randomized function, then we write $y \xleftarrow{\$} f(x)$ as evaluating f on input x with uniform random coins producing y; when we need to explicitly specify the random coins, we write $y = f(x; r)$. The statistical distance between two discrete probability distributions X and Y is defined as $\|X - Y\| = \frac{1}{2}\Sigma_z |\Pr[X = z] - \Pr[Y = z]|$. Let $\mathsf{Bernoulli}(p)$ denote the Bernoulli distribution with success probability p. Let x be a binary string, we denote by $\mathcal{D}_\varepsilon(x)$ the distribution over $\{0,1\}^{|x|}$ produced by the random process of sampling a string x' by flipping each bit of x independently with probability ε.

3 A Complexity Measure for Correlated-Instance

One of the goals of research in the design and analysis of algorithms is to develop algorithms which work well in practice taking into account all available data. We have demonstrated that access to multiple problem instances with correlated solutions in the CSP domain, can change the complexity of problems from intractable to efficiently solvable. We believe that this research direction should be explored further, for other problems and algorithms.

Our results lead us to propose a *correlated* complexity measure for search problems. Correlated-instance complexity measures the performance of an algorithm on a worst case (or average case) primary instance given as auxiliary input a tuple of correlated instances where the correlation is over the solution space.

Briefly, let R be an $\mathcal{NP}$ relation consisting of over pairs (I, S) of instance I and solution S. Let $\mathrm{R}_n \subset \mathrm{R}$ be set of pairs (I, S) with instance size $|I| = n$.

In the correlated-instance setting a problem is specified by a triple $P = (\mathrm{R}, \mathcal{D}, corr\mathcal{D})$ where R is an $\mathcal{NP}$ relation, $\mathcal{D}$ is a family of distributions $\mathcal{D}_n$ called the primary-instance distributions, and $corr\mathcal{D}$ is a family of distributions $corr\mathcal{D}(I, S; T)_n$ parameterized by $(I, S) \in \mathrm{R}_n$ and a function $T(n)$, which we call the correlated-instance distributions. The distribution $\mathcal{D}_n$ is over pairs $(I, S) \in \mathrm{R}_n$, and the distribution $corr\mathcal{D}(I, S; T)_n$ is over tuples of pairs $(\vec{I}, \vec{S}) \in (\mathrm{R}_n)^T$ (here n is implicitly given through I).

An algorithm A is said to *solve the problem* $P = (\mathrm{R}, \mathcal{D}, corr\mathcal{D})$ *with* T *samples* if for every n, and every instance $(I, S) \in \mathrm{R}_n$, and for every $T(n)$-tuple drawn from the distribution $corr\mathcal{D}(I, S; T)_n$, the algorithm on input $I, \vec{I}$ outputs a solution S' such that $(I, \tilde{S}) \in \mathrm{R}_n$. A has correlated-instance worst-case complexity of $g(n)$ if

$$\max_{(I,S)\in \mathrm{R}_n} \mathop{\mathbb{E}}_{(\vec{I},\vec{S})} \left[\mathcal{C}_A(I, \vec{I})\right] = g(n)$$

where $(\vec{I}, \vec{S})$ is a T-tuple sampled from $corr\mathcal{D}(I, S; T)_n$. A has correlated-instance average-case complexity of $g(n)$ if

$$\mathop{\mathbb{E}}_{(I,S)} \mathop{\mathbb{E}}_{(\vec{I},\vec{S})} \left[\mathcal{C}_A(I, \vec{I})\right] = g(n)$$

where (I, S) is sampled from $\mathcal{D}_n$ and then $(\vec{I}, \vec{S})$ is a T-tuple sampled from $corr\mathcal{D}(I, S; T)_n$.

This formalism may be viewed in the context of a long line of works that try to find a bridge between worst case complexity and average case complexity. Worst case is successfully analyzed in theory, but does not reflect the empirical nature of instances. On the other hand, it is very hard to describe the distributions that occur in practice. This has lead to a line of works that try to find a model blending worst case and average case. Smoothed analysis provides a very successful such attempt, yet on discrete instances this seems more difficult. Our formalism suggests a new blend between average case and worst case.

We note that extrapolating from the CSP results, if for a search problem, a (natural or forced) underlying generating process is available of instances of correlated solutions, one would be well advised to attempt the following two-step algorithm design strategy: first, derive the "difference" between the solutions of the generated instances; then use these differences to find the solution.

Comparison with smooth analysis: The celebrated work on smooth analysis by Spielman and Teng[ST04] introduced a new measure on the complexity of algorithms, first illustrated via the smooth analysis of the celebrated Simplex algorithm in the realm of real valued inputs. Smooth

analysis measure for an input x , is the expected behavior of the algorithm on correlated inputs which are the result of subjecting x to perturbations (e.g. flip its bits with a certain probability). The correlated instance complexity measure we introduce, is different in several aspects. First, we don't consider inputs which correlated to a primary input x, but rather inputs whose underling solutions correlated to the solution of x. Second, we deviate from the traditional paradigm of an algorithm working on a single input x, toward the design of algorithms which receive input x as well as in addition auxiliary inputs x_i's whose solutions are correlated to the solution to x, to assist in the goal of computing the solution for x.

We mention that Spielman and Teng [ST03] in a follow up work to their original smooth analysis paper observe that in the discrete input domain, perturbations of the input "should probably be restricted to those which preserve the most significant aspect input with respect to a given situation". To address this, they define property-preserving perturbations to inputs and relate this measure to property testing work [GGR96] and the heuristics of Feige and Kilian [FK01] for finding cliques on semi-random graphs with planted cliques.

4 CSP with Correlated Instances

In this section, we demonstrate the benefits of correlation in the context of CSP, by considering a natural distribution where multiple instances have ε-correlated solutions. We start by reviewing the classical CSP and then introducing the problems with correlated instances.

4.1 Definitions

CSP A CSP graph is a mapping $G : [m] \times [d] \to [n]$. G can be viewed as indicating the d neighbors in $[n]$ of each $j \in [m]$, so it is also conveniently depicted as a bipartite graph on "input" vertexes $[n]$ and "output" vertexes $[m]$ such that each $j \in [m]$ is connected to d neighbors in $[n]$: $G(j,1), \ldots, G(j,d)$. For brevity of notations, we also denote by $G(j)$ the list of all neighbors $G(j,1), \cdots, G(j,d)$ of output vet ice j. A CSP predicate is simply $P : \{0,1\}^d \to \{0,1\}$, and d is the arity of P. Given G and P we define the mapping $f_{G,P} = P \circ G : \{0,1\}^n \to \{0,1\}^m$ taking an input $x \in \{0,1\}^n$ to an output $y \in \{0,1\}^m$ by

$$\forall j \in [m] \quad y_j = P(x_{G(j)})$$

The classical worst-case CSP problem with predicate P and density D asks that, given a graph G and an output y, how to find an input x that satisfies all constraints.

Definition 4.1 (Worst-case-CSP(P, D))**.** *Let (G, y) be any graph $G : [Dn] \times [d] \to [n]$ and image $y = P \circ G(x)$ for any $x \in \{0,1\}^n$. Find a solution $\tilde{x}$ consistent with y, that is, $P \circ G(\tilde{x}) = y$.*

The average-case CSP asks how to solve the same problem for a random graph G and output y of a random hidden input.

Definition 4.2 (Average-case-CSP(P, D))**.** *Let (G, y) be generated by selecting at random a graph $G : [Dn] \times [d] \to [n]$ and a string $x \in \{0,1\}^n$, and outputting G and $y = P \circ G(x)$. Find a solution $\tilde{x}$ consistent with y.*

Correlated CSP We consider a *correlated-instance setting* in which, in addition to (G, y), multiple *auxiliary* instances $(G^1, y^1), \ldots (G^T, y^T)$ correlated with (G, y) are available; the algorithmic goal is still to find a solution x to (G, y), referred to as the *primary* instance, or even to find solutions $x, x^1, \cdots, x^T$ for all instances $y, y^1, \cdots, y^T$. We investigate a specific correlation between the auxiliary and primary instances, where all auxiliary instances share the same graph as the primary instance, that is, $G^t = G$, and the hidden input x^t of each auxiliary instance is a ε-noisy copy of the hidden input x of the primary instance. More precisely, let $\mathrm{coCSP}_\varepsilon(P, G, x)$ be a distribution that samples an instance correlated with (G, x, y) by selecting at random $x' \xleftarrow{\$} \mathcal{D}_\varepsilon(x)$, computing $y' = P \circ G(x')$ and outputting (G, x', y').

Similar to the classical setting, one can consider either a worst-case primary instance or an average-case one. Worst-case (ε, T)-correlated CSP with predicate P and density D considers the following question:

Definition 4.3 (Worst-case (ε, T)-correlated-CSP(P, D))**.** *Let the primary instance (G^0, y^0) be any graph $G^0 : [Dn] \times [d] \to [n]$ and image $y^0 = P \circ G^0(x^0)$ for any $x^0 \in \{0,1\}^n$. Choose T auxiliary instances $(G^1, y^1), \ldots, (G^T, y^T)$ by selecting at random $(G^t, x^t, y^t) \xleftarrow{\$} \mathrm{coCSP}(P, G^0, x^0)$ for all $t \in [T]$. Given $\left\{(G^t, y^t)\right\}_{0 \le t \le T}$, find solution $\tilde{x}^0$ consistent with y^0.*

We say that an algorithm solves worst-case (ε, T)-correlated-CSP(P, D)) with probability $p(n)$, if for every primary instance, the algorithm finds a solution with probability $p(n)$ over the randomness for generating the auxiliary instances.

The average-case (ε, T)-correlated CSP with predicate P and density D considers the same question for a randomly generated primary instances.

Definition 4.4 (Average-case (ε, T)-correlated-CSP(P, D))**.** *Choose the primary instance (G^0, y^0) by selecting at random a graph $G^0 : [Dn] \times [d] \to [n]$ and a string $x^0 \in \{0,1\}^n$, and outputting G^0 and $y^0 = P \circ G^0(x^0)$. Choose T auxiliary instances $(G^1, y^1), \ldots, (G^T, y^T)$ by selecting at random $(G^t, x^t, y^t) \xleftarrow{\$} \mathrm{coCSP}(P, G^0, x^0)$ for all $t \in [T]$. Given $\left\{(G^t, y^t)\right\}_{0 \le t \le T}$, find solution $\tilde{x}^0$ consistent with y^0.*

We say that an algorithm solves average-case (ε, T)-correlated-CSP(P, D)) with probability $p(n)$, if the algorithm finds a solution to the a randomly generated primary instance with probability $p(n)$, over the randomness for generating all instances.

Approximation In the classical setting, an $(1 - \alpha)$-approximation algorithm of the (worst-case or average-case) CSP returns a solution that satisfies at least a $(1 - \alpha)$ fraction of the constraints in the CSP instance. Similarly, an $(1 - \alpha)$-approximation algorithm of the (worst-case and average-case) correlated CSP returns a solution $\tilde{x}^0$ that satisfies at least a $(1 - \alpha)$ fraction of the constraints in the primary instance.

Correlated CSP with Differences We also consider a relaxed correlated-instance setting, where in addition to the auxiliary instances, the differences $x^t \oplus x^0$ between their hidden inputs x^t and the hidden input x^0 of the primary instance is known. Such a relaxed setting captures, for instance, the scenario where the correlation between instances is created via tampering. The algorithmic goal is still finding a solution to the primary instance. Formally,

Definition 4.5 (Worst-case/Average-case (ε, T)-correlated-CSP(P, D) with differences)**.** *Let (G^0, x^0, y^0) be a worst-case instance as defined in Definition 4.3 or an average-case instance sampled at random*

as in Definition 4.4. For every $t \in [T]$, let (G^t, x^t, y^t) be sampled at random as in Definition 4.3 and 4.4.

Given (G^0, y^0) and $\{(G^t, y^t, \Delta^t)\}_{t\in[T]}$, where $\Delta^t = x^t \oplus x^0$, find solution $\tilde{x}^0$ consistent with y^0.

4.2 Our Results

We start with designing algorithms in the simpler setting of correlated-CSP with differences and then try to extend the algorithms to remove the dependence on knowing differences.

Algorithms for Correlated-CSP with Differences We start with the simpler setting of correlated CSP with differences. In this relaxed setting, we first design an algorithms for solving *worst-case* (ε, T)-correlated-CSP(P, D), for a generic $\varepsilon \in (0, 1/2)$, any non-trivial (i.e., non-constant) predicate P and density D, assuming that the number of auxiliary instances is sufficiently large. Here, the number of auxiliary instances affects the the success probability of the algorithms; in particular, for any constant $r > 0$, if a logarithmic number $O(\log n)$ of auxiliary instances are available, the algorithms solves the worst-case primary instance with $1 - n^{-r}$ probability. Formally,

Theorem 4.1. *For every $\varepsilon \in (0, 1/2]$, there is a polynomial time algorithm $\mathcal{C}$, such that for every predicate $P : \{0,1\}^d \to \{0,1\}$, density $D \geq 1$, and constant $r > 0$, $\mathcal{C}$ solves worst-case (ε, T)-correlated-CSP(P, D) with differences, with success probability $1 - n^{-r}$, provided that $T > 8(d + \log(Dn^{r+1}))/\varepsilon^d$.*

In fact, since the above algorithm $\mathcal{C}$ handles any predicate P, it can be easily extended to solving more general CSPs where every constraint is with respect to a different predicate $P_1, \ldots, P_m$, $m = Dn$. Finding a solution to worst-case instances of such general CSPs is equivalent to inverting arbitrary $\mathcal{NC}^0$ computation. We modify the algorithm $\mathcal{C}$ to invert Boolean functions with bounded output-locality d, given a sufficiently large number $T = \Omega((\log n)2^{2d})$ of correlated instances as described in the theorem below.

Corollary 4.1. *For every $\varepsilon \in (0, 1/2]$, there is an algorithm satisfying that, for every Boolean function $f : \{0,1\}^n \to \{0,1\}^m$ with output locality d, string $x \in \{0,1\}^n$, and constant $r > 0$, the algorithm on input f, $y = f(x)$, $y^t = f(x^t)$ and $\Delta^t = x \oplus x^t$ where $x_t \xleftarrow{\$} \mathcal{D}_\varepsilon(x)$ for every $t \in T$, inverts y with probability $1 - n^{-r}$ in $O(mn^2T2^{3d})$ steps, provided that $T > 8(d + \log(Dn^{r+1}))/\varepsilon^d$.*

If f can be computed in $\mathcal{NC}^0$, the output locality d is a constant, and thus the above theorem establishes a polynomial time inverting algorithm for f, requiring a logarithmic number of correlated samples. If f has a logarithmic output locality $d = O(\log n)$, the above theorem gives a polynomial time inverting algorithm requiring a polynomial number of correlated samples.

Theorem 4.1 and Corollary 4.1 demonstrates the power of having many correlated instances while knowing the differences between the hidden inputs. A question that arises immediately is "does correlation help when only a few auxiliary instances are available?" Towards understanding this question, we show a second algorithm in the relaxed setting (where the differences of hidden inputs are known) that works with only a single auxiliary instance. However, this algorithm does not handle generic predicate and parameters as the algorithm in Theorem 4.1 does. Instead, it only handles a large class $\mathcal{P}$ of predicate $P(a)$ satisfying that one of its derivative $P^\sigma(a)$ is γ-correlated with one of the input bits a_{i^*}. Furthermore, it requires the flipping probability p^* and density D to lie within certain range. More precisely,

Theorem 4.2. *Let $P(a)$ be any predicate satisfying that there exist $\sigma \in \{0,1\}^d$, $\gamma > 0$ and $i^* \in [d]$, such that, the derivative $P^\sigma(a)$ is γ-correlated with the i^{*th} input bit a_{i^*}. For any $\varepsilon \in (0, 1/2)$, let the flipping probability p^* be set as*

$$p^* = \begin{cases} \varepsilon & \text{if } \sigma_t = 0 \\ 1-\varepsilon & \text{else } \sigma_t = 1 \end{cases}$$

Assume that for a sufficiently large constant N,

$$\varepsilon \leq 1/N2^{2d}d^6 \qquad \text{and} \qquad D \geq N/\varepsilon^{5d}.$$

For every $r \geq 1$, there is a polynomial time algorithm $\mathcal{A}$ that solves average-case $(p^*, 2)$-correlated-CSP(P, D) with differences, with probability $1 - O(n^{-r})$.

In fact, depending on P and P^σ, much weaker bounds on ε and D suffice: Let β be the boundary of P, and κ the probability that a d-bit string a, whose each bit is independently Bernoulli distributed with success probability p^*, matches σ at all bits but the $i^{*\text{th}}$. ε only needs to be smaller than β^2/Nd^6 and D needs to be greater than $N(d^8/\beta^2 + \ln(1/\varepsilon)/\gamma^2\kappa^2)$. The more precise ranges of ε and D can be found in Proposition 4.1.

Algorithms for Correlated-CSP without Differences Next, we establish two theorems similar to Theorem 4.1 and 4.2 in the more stringent setting where the differences between the hidden inputs are not known. To this end, we design two algorithms that given the primary and auxiliary instances estimate the differences between their hidden inputs; then, to solve the primary instance, we can simply apply algorithms in Theorem 4.1 and 4.2 using the estimated differences. Our first difference estimation algorithm works with a randomly chosen primary instance (leading to an algorithm for the average-case correlated CSP) and for a restricted range of parameters when ε is sufficiently small (with respect to d) and the density D is sufficiently large (with respect to d and ε). In these settings, given a pair of primary and auxiliary instances, the algorithm outputs an estimated difference between their hidden inputs that is guaranteed to be α-close to the actual difference for an arbitrarily small constant α. Combining this algorithm with Theorem 4.2 gives the following theorem:

Theorem 4.3. *Let $P(a)$ be any predicate satisfying that there exist $\sigma \in \{0,1\}^d$, $\gamma > 0$ and $i^* \in [d]$, such that, the derivative $P^\sigma(a)$ is γ-correlated with the i^{*th} input bit a_{i^*} and $\sigma_{i^*} = 0$. Assume that for some sufficiently large constant N,*

$$\varepsilon \leq 1/N2^{3d}d^6 \qquad \text{and} \qquad D \geq N/\varepsilon^{5d}.$$

For every $r \geq 1$, there is a polynomial time algorithm $\mathcal{A}$ that solves average-case $(\varepsilon, 2)$-correlated-CSP(P, D), with probability $1 - O(n^{-r})$.

As Theorem 4.2, much weaker bounds on ε and D suffices, depending on properties of P, and P^σ: Let β, κ be defined as above, and let ρ be the maximum influence of different input bits of P, $\rho = \max(\text{Inf}_P(i))$; then it suffices to have $\varepsilon \leq \rho\beta^2/Md^6$ and $D \geq Md^8(1/\beta^2 + 1/\rho^2 + 1/\gamma^2\kappa^2)\ln(1/\gamma\kappa\varepsilon)$, for some sufficiently large constant M. The more precise bounds can be found in Proposition 4.3.

Towards establishing an analogy of Theorem 4.1, the first difference estimation algorithm is, however, insufficient, for the reason that though the estimated differences is correct for most (a $(1-\alpha)$ fraction of) input bits, it is not known where the errors are. We thus design a second difference estimation algorithm that utilizes multiple auxiliary instances to identify an α fraction

of the input bits, for which the error probability is high, and outputs estimated differences for the rest input bits for which the error probability is low and can be tolerated. Thus combining the second difference estimation algorithm with Theorem 4.1 we obtain,

Theorem 4.4. *Fix any non-trivial predicate P and constant $\alpha^* \in (0,1]$. Assume that for a sufficiently large constant N,*

$$\varepsilon < 1/4d2^d, \qquad D > \frac{Nd}{\varepsilon^2\alpha^*}\log\left(\frac{2}{\varepsilon}\right), \qquad \text{and } T = T(n) \geq Nd^2\log(Dn)/\varepsilon^{2d}.$$

There is a polynomial time algorithm $\mathcal{B}$ that finds $(1-\alpha^*)$-approximate solutions for average-case (ε, T)-correlated-CSP(P, D) with probability $1 - 3/n$, for sufficiently large $n \in \mathbb{N}$.

Again, a weaker bound of ε, depending on the maximum influence ρ of P suffices. See Proposition 4.4 for more details.

4.3 Overview of Techniques

We give an overview of our techniques for proving the above four theorem. Some part of this overview repeats that in Section 1.2 in the introduction, but more details are provided.

Solving multiple correlated CSPs with difference We start with the most relaxed setting, where to solve a primary instance G, y, multiple auxiliary instances $(G, y^1, \cdots, y^T)$ are available and the differences $(\Delta^1, \cdots, \Delta^T)$ between their hidden inputs $(x^1, \cdots, x^T)$ and that x of the primary instance are known. In this setting, we design an algorithm $\mathcal{C}$ that is able to solve any worst-case primary instance, for generic predicate P, density D and setting of the correlation parameter $\varepsilon \in (0, 1/2]$.

To illustrate the idea behind $\mathcal{C}$ for finding a solution $\tilde{x}$ consistent with y, assume that an *infinite* number of auxiliary instances are available $y^1, y^2, \cdots$ together with the corresponding differences $\Delta^1, \Delta^2, \cdots$ between their hidden inputs and x. Then, consider an output vertex $k \in [m]$: Its value in the t^{th} instance is

$$y_k^t = P(x_{G(k)}^t) = P(x_{G(k)} \oplus \Delta_{G(k)}^t)$$

For every instance t, the difference $\Delta_{G(k)}^t$ (restricted to the neighbors of k) is a d-bit string, where every bit $\Delta_{G(k,l)}^t$ is independently Bernoulli distributed with probability ε of being 1. Given an infinite number of auxiliary instances, every possible "shift" $a \in \{0,1\}^d$ will appear (i.e., for every $a \in \{0,1\}^d$, $\exists t$ s.t. $\Delta_{G(k)}^t = a$). Since the differences $\{\Delta_{G(k)}^t\}$ are known, the algorithm $\mathcal{C}$ can collect the values of P on every possible shift a from $x_{G(k)}$, which form an encoding of $x_{G(k)}$; we call it the *shift-P-encoding* of $x_{G(k)}$. More specifically,

$$\text{Encode}(z) \triangleq \left[P(z \oplus \underbrace{0\cdots0}_{\text{d}}), P(z \oplus \underbrace{0\cdots0}_{\text{d-1}}1), \cdots, P(z \oplus \underbrace{1\cdots1}_{\text{d}})\right] \in \{0,1\}^{2^d}$$

Given encoding $\text{Encode}(x_{G(k)})$, if the predicate P satisfies that the encoding for every $z \in \{0,1\}^d$ is unique, then we can uniquely recover $x_{G(k)}$. (This can be done efficiently, since the encoding has constant size 2^d.) Then using the same procedure for every output bit k, the algorithm uncovers the unique value of $x_{G(k)}$ for every k. By stitching them together, it obtains the unique string x consistent with y as well as all $\{\Delta^t\}$ and $\{y^t\}$.

To extend the above intuitive idea to the full-fledged algorithm, we first modify the algorithm to work with a logarithmic number $T = O(\log n)$ number of auxiliary instances. It is easy to see that, for a constant locality d, when T is sufficiently large (still logarithmic), the encoding $\mathrm{Encode}(x_{G(k)})$ for every input bit k can be recovered completely with high probability $1 - n^{-r}$; then the algorithm proceeds as before.

The real challenge is *how to go beyond predicates that admit unique encoding and handle all nontrivial predicates*. The difficulty lies in that for a general predicate P, there may exist different strings $z \neq z'$ that have the same encoding $\mathrm{Encode}(z) = \mathrm{Encode}(z')$. Then, from the encoding $\mathrm{Encode}(x_{G(k)})$, the algorithm can only deduce that there are multiple candidates for the value of $x_{G(k)}$. However, given a collection of candidates for all sub-strings $\{x_{G(k)}\}$, it is not clear how to find a single string $\tilde{x}$ that simultaneously satisfies all constraints on all sub-strings $\{x_{G(k)}\}$.

We overcome this problem by showing that given a valid encoding E (for which there exists a pre-image), all the pre-images of E form a unique affine space Λ. Then, given encoding $\mathrm{Encode}(x_{G(k)})$, the constraints on $x_{G(k)}$—that is, $\{P(x_{G(k)} \oplus \Delta^t_{G(k)}) = y^t_k\}_t$—is equivalent to a set of linear constraints on $x_{G(k)}$, that is, $x_{G(k)} \in \Lambda$. Now, given the collection of linear constraints for all k, the algorithm can solve the linear system efficiently to recover a string $\tilde{x}$ consistent with all constraints.

It is easy to see that this method goes through even if different output bits are evaluated using different predicates, leading to an algorithm for inverting any $\mathcal{NC}^0$ functions. In fact, in its full generality, this method can be used to invert any function with output locality d given a sufficiently many $T = O(\log n/\varepsilon^d)$ auxiliary instances.

Solving two correlated CSPs with difference When only a single auxiliary instance is available, we cannot hope to collect enough statistical information about each output bit as discussed above. Instead, if the CSP instances have *high density* D, then with very high probability over the randomly choice of the primary instance graph G, most input bits participates in the calculation of multiple output bits. Thus, we can collect statistical information about every input bit directly. Below we describe how to use this method to solve average-case $(G, y), (G, y')$, provided that the predicate $P(a)$ has the property that one of its derivative $P^\sigma(a)$ is correlated with one of its input bits a_{i^*}.

At a high-level, for any input bit i, the algorithm $\mathcal{A}$ tries to collect statistical information about the value of x_i from all the output bits k that have i as the $i^{*\text{th}}$ input bit (i.e., $G(k, i^*) = i$); we call such an output bit a t-neighbor of i. If it occurs that the differences $\Delta_{G(k)} = \sigma$, it holds that

$$y_k \oplus y'_k = P(x_{G(k)}) \oplus P(x_{G(k)} \oplus \Delta_{G(k)}) = P(x_{G(k)}) \oplus P(x_{G(k)} \oplus \sigma) = P^\sigma(x_{G(k)})$$

Therefore, when $\Delta_{G(k)} = \sigma$ occurs, $\mathcal{A}$ can efficiently compute $P^\sigma(x_{G(k)}) = y_k \oplus y'_k$, and by the fact that P^σ is positively correlated with the $i^{*\text{th}}$ input bit, $\mathcal{A}$ can use $P^\sigma(x_{G(k)})$ as a vote to the value of $x_i = x_{G(k,i^*)}$. Thus, to estimate x_i, $\mathcal{A}$ collects all votes from i's i^*-neighbors satisfying $\Delta_{G(k)} = \sigma$, and check whether a large enough fraction of them vote for 1. We show that as long as *sufficiently many* votes are collected from i's t-neighbors, the voting scheme yields a correct estimation $\tilde{x}_i = x_i$ with high probability ($\geq (1 - \beta)$ for any $\beta > 0$). Furthermore, since the estimation for each i only depends on "local information" related to its i^*-neighbors, via a careful analysis of the randomized procedure for generating the pair of instances, we show that each estimation is independent of others; thus, by Chernoff bound, a large fraction of these input bits *that have enough votes* are estimated correctly. (We remark that the argument of independence relies crucially on the fact that the graph G is random chosen; in a random graph, the choices of neighbors of every output bit k and the values of neighbors are independent, conditioned on the weights of Δ and x.)

Now it only remains to argue that for most input bits, enough votes are collected. We rely on the fact that the instances have high density to argue that most input bits have many t-neighbors; however, this is not sufficient. Consider an input i and one of its i^*-neighbor k, for the event $\Delta_{G(k)} = \sigma$ to occur, it must be that $\Delta_i = \Delta_{G(k,i^*)} = \sigma_{i^*}$; thus, input bits that have $\Delta_i \neq \sigma_{i^*}$ would never collect any votes. This puts an interesting requirement on the correlation between x and x' depending on the predicate P. Namely, if $\sigma_{i^*} = 0$, the correlation between x and x' needs to be sufficiently large so that most input bits i have $\Delta_i = 0$ and thus its value can be estimated using the voting strategy. On the other hand, if $\sigma_{i^*} = 1$, the correlation needs to be sufficiently small so that most input bits i have $\Delta_i = 1$. If the auxiliary instance is appropriately correlated with the primary instance, the algorithm $\mathcal{A}$ can indeed collect enough votes for most input bits, and derive correct estimation for most of them, leading to an "almost-correct" solution $\tilde{x}$ that is $(1-\beta')$-close to the hidden input x. Finally, we apply the procedure designed by Bogdanov and Qiao in [BQ12], to turn an almost-correct solution $\tilde{x}$ into a true solution.

We remark that, unlike the algorithm $\mathcal{C}$ which solves worst-case primary instances, the algorithm $\mathcal{A}$ only solves an average-case primary instance; this is because the analysis of $\mathcal{A}$ crucially relies on the fact that the graph G and the primary hidden input x are chosen at random to argue that G and x are "well formed"—with properties such as the number of t-neighbors of most input bits are close to D, and x is relatively balanced, etc.—and that every t-neighbor of an input bit i has a constant probability of generating a correct vote, independent of each other. One can potentially characterize these (G, x) for which a solution can be found; however this is outside the scope of this work, and we leave this as future work.

Solving two correlated CSPs without difference: In the more stringent setting where the differences between the hidden inputs are not known, we design two algorithms analogous to the algorithms $\mathcal{C}$ and $\mathcal{A}$ described above. Ideally, to remove the dependence on the differences, we want to design sub-algorithms that given the primary and a single or multiple auxiliary instances estimate the differences between the hidden inputs; then we can simply invoke the algorithms $\mathcal{C}$ and $\mathcal{A}$ with the estimated differences. However, two algorithmic challenges exist: (1) How to estimate the differences with potentially small errors? (2) Are algorithms $\mathcal{C}$ and $\mathcal{A}$ robust to small noises in the differences they receive? and if not, how to adapt them to become robust? Below we start with the simpler case where only a single auxiliary instance is available, and describe the intuition on how to overcome these challenges in this case.

Assume that the predicate P has an input bit, say the ℓ^{th}, with high influence, that is, when the ℓ^{th} input bit flips, with probability ρ over the uniform random choices of the other $d-1$ input bits, the output of the predicate also flips (i.e., $\rho = \Pr_{x \leftarrow U_d}[P(x) \oplus P(x^{\oplus \ell})]$ where $x^{\oplus \ell}$ is x with the ℓ^{th} bit flipped); this assumption is without loss of generality, as any non-trivial predicate P must have an input bit with influence at least 2^{-d}. In other words, when the rest $d-1$ input bits remain the same, whether the output flips or not is positively correlated with whether the ℓ^{th} input bit flips or not.

Then, given a pair of correlated instances $(G, y), (G, y')$ generated at random, we can estimate whether an input bit i flips or not by collecting votes from its ℓ-neighbors: For every ℓ-neighbor k of i, vote that i flips or not depending on whether k flips or not respectively; if a large enough fraction of i's ℓ-neighbors vote that i flips, estimate that i flips (set $\tilde{Delta}_i = 1$), and otherwise, estimate that i remains the same (set $\tilde{Delta}_i = 0$). We argue that when the primary instance (G, y) is chosen at random, the correlation parameter ε is sufficiently small and density D is sufficiently high, the estimation $\tilde{Delta}$ is "almost correct" (i.e., $(1-\beta)$-close to the actual difference Δ) with high probability.

When ε is sufficiently small, x and x' differ at a small fraction (close to ε) of the input bits. Therefore, in a random graph where k's $d-1$ non-ℓ-neighbors are chosen uniformly randomly, with high probability (close to $(1-\varepsilon)^{d-1}$), all non-ℓ-neighbors of k do not flip. In this case, by the fact that ℓ^{th} input bit of P has high influence, whether $y_k \neq y'_k$ is positively correlated with whether $x_i \neq x'_i$. Thus the vote from k is correct with some constant probability. Then, by a similar argument as that for algorithm $\mathcal{A}$, with high probability a correct estimation is derived for every input bit that has enough votes, and all estimation are independent. Then, given that the density is high, in a random graph, most input bits have many ℓ neighbors and thus enough votes, leading to the conclusion that the estimation is almost correct.

The next question is whether the algorithm $\mathcal{A}$ is robust to an almost correct difference $\tilde{D}elta$. Recall that $\mathcal{A}$ uses a voting scheme to guess the value x_i of each input bit, depending on the local information related to its t-neighbors. The voting scheme can be naturally extended to tolerate a small amount of errors in the neighborhood of its t-neighbors. Thus, when the total fraction of errors is bounded, the fraction of input bits i for which there are too many errors in the neighborhood of its t-neighbors is also bounded by a constant. Therefore, $\mathcal{A}$ is robust.

Solving many correlated CSPs without difference: When multiple auxiliary instances $(G, y^1, \cdots, y^T)$ are available, we hope to retain the capability of handling general predicates. The first and straightforward approach is applying the above difference estimation algorithm $\mathcal{A}_\Delta$ to estimate the difference $\Delta^t = x \oplus x^t$ for each auxiliary instance y^t by invoking $\mathcal{A}_\Delta$ with $(G, y), (G, y^t)$ to obtain an estimated difference $\tilde{D}elta^t$. As argued above, for every t, $\tilde{D}elta^t$ is correct for all but a small constant fraction of the input bits, with high probability. We hope that the algorithm $\mathcal{C}$ is or can be extended to be robust against such noisy estimated differences $\tilde{D}elta^1, \cdots \tilde{D}elta^T$.

However, this approach does not go through for the following reason: Although the error rate in each estimated difference $\tilde{D}elta^t$ is bounded, the error rate in the estimation for each input bit i, $\{\Delta_i^t\}$, is not. (Think of the former as a row and the latter as a column). In fact it could happen that there is a constant fraction of input bits for which the error rate in $\{\tilde{D}elta_i^t\}$ is not bounded. Unfortunately, the algorithm $\mathcal{C}$ is not robust to such noises: Recall that $\mathcal{C}$ tries to construct the shift-P-encoding of the neighborhood $x_{G(k)}$ of each output bit k, from the output bit values $y_k^1, \cdots, y_k^T$ and the differences $\tilde{D}elta_{G(k)}^1, \cdots, \tilde{D}elta_{G(k)}^T$ on the neighborhood. If there is an input bit in the neighborhood $i \in G(k)$, whose estimated differences are mostly incorrect, the resulting encoding $\text{Encode}(x_{G(k)})$ is wrong, leading to a set of incorrect linear constraints on $x_{G(k)}$. Therefore, if the estimated differences are wrong consistently on a constant fraction of the input bits, $\mathcal{C}$ derives a linear equation system with a constant fraction of errors; it is well known that linear systems with a constant fraction of errors are potentially computationally intractable.

To overcome this, we design a new difference estimation algorithm $\mathcal{B}$. The key feature of the new algorithm is that it can identify these (constant fraction of) input bits for which the estimation error is not bounded; and for the rest, it outputs estimations with bounded errors. (We note that the algorithm $\mathcal{C}$ can be extended naturally to accommodate bounded errors in each coordinate $\{\tilde{D}elta_i^t\}$.) The intuition behind the new algorithm is as follows: Fix an input bit i. Our method for estimating the differences in the i^{th} input bit follows the rationale that when i flips, we expect to see a larger fraction of its ℓ-neighbors flip, than the case where i does not flip. To examine this condition, we say that i is "influential" to one of its ℓ-neighbor k, if the value of k flips when i flips and all other input bits of k remain the same. If ε is sufficiently small, and i is "influential" to a large fraction of its ℓ-neighbors k, then indeed the condition holds (that is, a larger fraction of i's ℓ-neighbors flip values when i flips). Note that whether i is influential to k is decided by the

choice of k's non-ℓ-neighbors and their values. Over the random choices of sampling G, x, with high probability in proportion to ρ, i is influential to k. However, with a small constant probability, it happens that i is not influential to most of its ℓ-neighbors, in which case the estimation for i has high error rate.

The algorithm $\mathcal{B}_\Delta$ tries to identify such input bits. To see that this is possible, consider two extreme cases: (i) An input bit i is not influential to any of its ℓ-neighbors, and (ii) i is influential to all of its ℓ-neighbors. Our analysis later shows that in the former case, the fraction of flips in i's ℓ-neighbors would be consistently low across T instances no matter whether i flips or not, whereas in the latter the fraction of flips would be high for a constant fraction of instances where i flips. Therefore, by examining the pattern of flips in i's ℓ-neighbors across T instances, we can collect statistical information about how influential i is to its ℓ-neighbors. This allows us to spot these input bits that would have high error rate, namely, ones whose ℓ-neighbors mostly do not flip across T instances; furthermore, for the rest input bits, the statistical information provides more accurate threshold τ_i for deciding whether i flips or not (instead of using a universal threshold for all input bits as in algorithm $\mathcal{A}_\Delta$ above).

Finally, since the fraction of input bits with high error rate is small, the final algorithm can simply ignore them and invoke (an extended variant of) $\mathcal{C}$ on the "sub-problem" induced by the rest of the input bits, for which reliable estimations is available, yielding an approximate solution that satisfies all but a small constant fraction of the output constraints.

4.4 Solving Many Correlated CSP Instances with Differences

Towards proving Theorem 4.1, we present an algorithm $\mathcal{C}$ for correlated CSP with differences for generic predicate P, density D and parameter ε, when a logarithmically many auxiliary instances is available. Then we extend the algorithm $\mathcal{C}$ to invert any function with low output locality, leading to Corollary 4.1.

Though the algorithm $\mathcal{C}$ requires knowing the exact differences between the hidden inputs of the auxiliary and primary instances, at the end of this section, we show that this requirement can be slightly relaxed and $\mathcal{C}$ can be extended to work with certain structured noisy differences. This extension will be very useful for later handling the more stringent scenario where the differences are hidden.

4.4.1 Construction and Analysis of the Algorithm $\mathcal{C}$ (Proof of Theorem 4.1)

On input a primary instance (G, y) and a logarithmic number $T = O(\log)$ of auxiliary instances $y^1, \cdots, y^T$ together with the corresponding differences $\Delta^1, \cdots, \Delta^T$, the algorithm $\mathcal{C}$ tries construct the shift-P-encoding of the neighborhood $x_{G(k)}$ of every output bit k.

$$\text{Encode}(x_{G(k)}) = \left[P(x_{G(k)} \oplus \underbrace{0 \cdots 0}_{\text{d}}), P(x_{G(k)} \oplus \underbrace{0 \cdots 0}_{\text{d-1}} 1), \cdots, P(x_{G(k)} \oplus \underbrace{1 \cdots 1}_{\text{d}}) \right] \in \{0,1\}^{2^d}$$

As we show below in Claim 4.1 that for every P, the set of pre-images of a valid encoding form an affine space; moreover, the space is a shift of a universal affine space. Then the algorithm $\mathcal{C}$ can map the encodings corresponding to each output bit k to a set of linear constraint on the neighborhood $x_{G(k)}$ of k. Thus finding a consistent solution $\tilde{x}$ simply reduces down to solving the linear system.

A technical caveat in the above intuition is that it implicitly assumes that every output bit k has d distinct neighbors; thus, given sufficiently many auxiliary instances, all possible shift a on the d input bits of k appear with high probability. However, given a worst-case graph G, it could

happen that some output bit k has repetitive neighbors, that is, $\exists l \neq l'$ such that $G(k,l) = G(k,l')$. In this case, no matter how many auxiliary instances are available, all shifts a on input bits of k have that $a_l = a_{l'}$. To overcome this, instead of considering k as output of P on d input bits, we can remove the repeating input bits and view k as the output of P' on $d-1$ input bits, where

$$P'(x_{G(k,1)}, \cdots, x_{G(k,l'-1)}, x_{G(k,l'+1)}, \cdots, x_{G(k,d)})) \\ = P(x_{G(k,1)}, \cdots, x_{G(k,l'-1)}, x_{G(k,l)}, x_{G(k,l'+1)}, \cdots, x_{G(k,d)})).$$

Then the same method above can be applied w.r.t. P' to discover the linear constraints over the $d-1$ non-repeating input bits of k.

For simplicity of exposition, below we first present our algorithm $\mathcal{C}$ and its analysis w.r.t. a graph G that does not have repeating neighbors for any output bit (that is, for every k and every $l \neq l' \in [d]$, $G(k,l) \neq G(k,l')$); then at the end of the section, we note in Remark 4.1 how to handle graphs with repeating neighbors.

Construction of $\mathcal{C}$:

Claim 4.1. *Fix any predicate $P : \{0,1\}^d \to \{0,1\}$. There is an affine space Λ_P, called the* invariant space of P, *satisfying that for every $E \in \{0,1\}^{2^d}$ such that there is a $z^* \in \{0,1\}^d$ with encoding $E = \mathrm{Encode}(z^*)$, the set of pre-images of E, $S = \{z \in \{0,1\}^d | \, \mathrm{Encode}(z) = E\}$, forms the affine space $\Lambda(E) = z^* + \Lambda_P$.*

Proof. We say that a value $a^* \in \{0,1\}^d$ is an *"invariant"* of a predicate P, if it satisfies that for all a, $P(a) = P(a + a^*)$. Every pair $z \neq z'$ that have the same encoding $\mathrm{Encode}(z) = \mathrm{Encode}(z')$ defines an invariant $a^* = z' - z$ of the predicate P.

$$\begin{array}{rcl} & & \mathrm{Encode}(z) = [P(z \oplus 0\cdots0), P(z \oplus 0\cdots01), \cdots, P(z \oplus 1\cdots1)] \\ & = & \mathrm{Encode}(z') = \big[P(z' \oplus 0\cdots0), P(z' \oplus 0\cdots01), \cdots, P(z' \oplus 1\cdots1)\big] \\ \implies & & \forall a \in \{0,1\}^d, P(a) = P(a + (z' - z)) \end{array}$$

Let Λ_P be the affine space spanned by all the invariants of P. We show that S forms the affine space $\Lambda(E) = z^* + \Lambda_P$.

1. $S \subseteq \Lambda(E)$**:** For every $z \in S$, $\mathrm{Encode}(z) = E = \mathrm{Encode}(z^*)$. Therefore, $z - z^*$ is an invariant and belongs to Λ_P. Thus $z \in z^* + \Lambda_P = \Lambda(E)$.

2. $\Lambda(E) \subseteq S$**:** For every $z \in \Lambda(E)$, $z = z^* + a$, where $a \in \Lambda_P$ is a linear combination of the invariants of P. As shifting z^* by any invariant does not change its encoding, $\mathrm{Encode}(z) = \mathrm{Encode}(z^*) = E$. Thus $z \in E$.

□

Given the claim, a formal description of algorithm $\mathcal{C}$ appears in Figure 1.

We analyze the time complexity of $\mathcal{C}$: It takes $O(2^{3d})$ steps to find the invariant space Λ_P using brute force by comparing the encoding of every pair of input strings. It is easy to see that the second and the third steps take $O(Tm2^d)$ and $O(m2^{2d})$ time. Finally, the complexity of solving a system of $m(d-d')$ linear equations with n unknowns is bounded by $O(n^2md)$. Overall, the algorithm runs in time μmn^2T2^{3d} for some universal constant μ.

$\mathcal{C}$ for solving worst-case CSP using many auxiliary instances and differences

Let P, D, ε be public parameters. On input $G, y, y^1, y^2, \ldots, y^T$, and the differences $\Delta^1, \ldots, \Delta^T$, proceed in four steps:

1. Find the invariant space Λ_P of P using brute force. Let $d' \leq d$ be the dimension of Λ_P.
2. For every output bit $k \in [m]$, try to recover the encoding $E_k = \mathrm{Encode}(x_{G(k)})$ associated with its corresponding input bits $x_{G(k)}$ as follows: For every $t \in [T]$, set the $(\Delta^t_{G(k)})^{\text{th}}$ entry in E_k to y^t_k.

   In the end, if there is an output bit $k \in [m]$ whose encoding E_k is incomplete (i.e., for some entry a, the value $P(x_{G(k)} \oplus a)$ is not set), output $\perp$.
3. For every output bit $k \in [m]$, find a pre-image z^* of E_k using brute force, which defines the affine space $\Lambda(E_k) = z^* + \Lambda_P$ that $x_{G(k)}$ lies in; write down the $d - d'$ linear equations enforcing that $x_{G(k)} \in \Lambda(E_k)$.
4. Finally, find one solution $\tilde{x}$ to the system of $m(d - d')$ linear equations. Output $\tilde{x}$.

Figure 1: The algorithm $\mathcal{C}$ finds a consistent solution for every CSP instance (G, x, y) with high probability, given a logarithmic number of auxiliary instances and information of differences between hidden inputs.

Analysis of $\mathcal{C}$: To show that $\mathcal{C}$ indeed solves the primary instance with probability $1 - n^{-r}$, we first show that for every output bit k and every string $a \in \{0,1\}^d$, a appears as the difference between $x^t_{G(k)}$ and $x_{G(k)}$ for $\Omega(T)$ times, with probability $1 - n^{-r}$.

Claim 4.2. *Let G be as described above. Fix any $x \in \{0,1\}^n$ and $y = P \circ G(x)$. Sample T auxiliary instances $(G, x^t, y^t) \xleftarrow{\$} \mathrm{coCSP}(P, G, x)$ independently. For any constant r, it holds that with probability $1 - n^{-r}$, for every output bit $k \in [m]$ and string $a \in \{0,1\}^d$, the number of instances $t \in [T]$ satisfying that $\Delta^t_{G(k)} = a$ is at least $(\varepsilon^d/2)T$, provided that $T > 8(d + \log(mn^r))/\varepsilon^d$.*

Proof. For every output bit k, every $a \in \{0,1\}^d$ and every instance $t \in [T]$, the probability that $\Delta^t_{G(k)} = a$ is at least ε^d. Since the differences $\Delta^t_{G(k)}$ are independently distributed across different instances, the probability that less than a $\varepsilon^d/2$ fraction of the instances have $\Delta^t_{G(k)} = a$—denote this Event as $\mathsf{Evt}_{k,a}$—is at most $\exp(-T\varepsilon^d/8)$ by the Chernoff bound, which is smaller than $1/(2^d mn^r)$ when $T > 8(d + \log(mn^r))/\varepsilon^d$. By a union bound, the probability that for the k^{th} bit, $\mathsf{Evt}_{k,a}$ occurs for any $a \in \{0,1\}^d$ is at most $1/mn^r$. Finally, by Markov inequality, the probability that there is an output bit $k \in [m]$ and that have $\mathsf{Evt}_{k,a}$ occurring for some a is at most n^{-r}. □

Given the above claim, we show that the algorithm $\mathcal{C}$ on input P, (G, x, y), $(G, x^t, y^t) \xleftarrow{\$} \mathrm{coCSP}(P, G, x)$ for $t \in [T]$, and $\{\Delta^t = x^0 \oplus x^t\}_{t \in [T]}$, finds $\tilde{x}$ consistent with y with probability $1 - n^{-r}$.

By Claim 4.2, when $T > 8(d + \log(mn^r))/\varepsilon^d$, it holds that for every output bit k and string $a \in \{0,1\}^d$, there is at least one instance t satisfying $\Delta^t_{G(k)} = a$, with probability $1 - n^{-r}$. In other words, with probability $1 - n^{-r}$, $\mathcal{C}$ recovers the encoding E_k for every output bit k completely. In this case, by Claim 4.1, for every output bit k, the constraints regarding $x_{G(k)}$—including $P(x_{G(k)}) = y_k$ and $\{P(x_{G(k)} \oplus \Delta^t_{G(k)}) = y^t_k\}_{t \in [T]}$—is equivalent to the linear constraints that $x_{G(k)} \in z^* + \Lambda_P$ defined by E_k. Then, solving the system of linear constraints collected for all output bits k yields a solution $\tilde{x}$ consistent with y, $\{y^t\}$ and $\{\Delta^t\}$.

By the analysis of the time-complexity of $\mathcal{C}$, when $d = O(1)$ $T = \Theta(\log(nm))$, $\mathcal{C}$ runs in polynomial time in n and m.

Remark 4.1. *The above presentation of $\mathcal{C}$ and analysis deals with a graph G that does not have repeating neighbors for any of its output bits. It is easy to modify $\mathcal{C}$ to also handle arbitrary graphs that have repeating neighbors for some of its output bits: For every output bit k that has repeating neighbors, view k as the output of a predicate P_k on less than d input bits with the repeating neighbors removed. This yields a graph G' whose output vertexes have different degrees (all no greater than d) and are computed using different predicates. Modify $\mathcal{C}$ to find the invariant space for every P_k and find the linear constraints for every output bit k in G' as described above. Finally, $\mathcal{C}$ solves the linear equation system to recover a solution. The same analysis as above goes through for this more general case. We omit the details here.*

4.4.2 Inverting Arbitrary Functions with Small Output-locality

It is easy to observe that the algorithm $\mathcal{C}$ above does not require that all constraints (for different output bits) are with respect to the same predicate P. In fact, we show that the algorithm can be easily modified to handle the more general case where every output bit k is computed via a different predicate P_k. In this more general case, solving a CSP instance (G, x, y), where $y_k = P_k(x_{G(k)})$ for every k, is equivalent to inverting a $\mathcal{NC}^0$ computation f, and the ideas behind the algorithm $\mathcal{C}$ can be directly applied to inverting any $\mathcal{NC}^0$ function in the worst case, given a logarithmic number of auxiliary instances and information of differences between hidden inputs. Below, we present this extension for inverting any function with low output locality.

The inverting algorithm $\mathcal{M}$ on input any function $f : \{0,1\}^n \to \{0,1\}^m$ with output locality d, any $f(x)$, and auxiliary instances $\{f(x_t)\}_{t\in[T]}$ whose pre-images are ε-noisy copies of x, proceeds as follows: Represent f as graph G and predicates $P_1, \ldots, P_m$ so that $f(x)_k = P_k(x_{G(k)})$; then proceed as $\mathcal{C}$ does except that in the first step, $\mathcal{M}$ finds the invariant spaces Λ_{P_k} for every predicate P_k, and correspondingly in the third step, for every output bit k, it finds the linear constraints on $x_{G(k)}$ w.r.t. predicate P_k as $x_{G(k)} \in z^* + \Lambda_{P_k}$. We provide a formal description of the modified algorithm $\mathcal{M}$ in Figure 2.

$\mathcal{M}$ for Inverting Functions with Small Output-Locality

On input $f : \{0,1\}^n \to \{0,1\}^m$, $y = f(x)$, $\{y^t = f(x^t)\}_{t\in[T]}$ and $\{\Delta^t = x^t \oplus x\}_{t\in[T]}$, proceed in following steps:

1. Represent f as graph G and predicates $P_1, \ldots, P_m$ so that $f(x)_k = P_k(x_{G(k)})$.
2. For every k, Find the invariant space Λ_{P_k} of P_k using brute force. Let $d'_{P_k} \leq d$ be the dimension of Λ_{P_k}.
3. For every output bit $k \in [m]$, try to recover the encoding $E_k = \mathrm{Encode}_{P_k}(x_{G(k)})$ associated with its corresponding input bits $x_{G(k)}$ as follows: For every $t \in [T]$, set the $(\Delta^t_{G(k)})^{\text{th}}$ entry in E_k to y^t_k.

   In the end, if there is an output bit $k \in [m]$ such that its corresponding encoding E_k is incomplete, output $\perp$.
4. For every output bit $k \in [m]$, find a pre-image z^* of E_k w.r.t. P_k using brute force, and write down the $d - d'_{P_k}$ linear equations with unknown $x_{G(k)}$ enforcing that $x_{G(k)} \in \Lambda_{P_k}(E_k) = z^* + \Lambda_{P_k}$.
5. Finally, find one solution $\tilde{x}$ to the system of at most md linear equations. Output $\tilde{x}$.

Figure 2: The algorithm $\mathcal{M}$ inverts any function with small output locality in the worst-case, given sufficiently many instances and information about the differences between hidden inputs.

TIME COMPLEXITY OF $\mathcal{M}$: Similar to the running time analysis of $\mathcal{C}$, it takes $O(m2^{3d})$ steps to find the invariant spaces Λ_{P_k} for each output bit k using brute force. The third and fourth steps (corresponding to the second and third steps in $\mathcal{C}$) take time $O(Tm2^d)$ and $O(m2^{2d})$ respectively. Finally, the complexity of solving a system of at most md linear equations with n unknowns is bounded by $O(n^2md)$. Overall, the algorithm runs in time μmn^2T2^{3d} for a universal constant μ.

Finally, the correctness of $\mathcal{M}$ follows syntactically the same proof of the correctness of $\mathcal{C}$ (proof of Theorem 4.1). This concludes Corollary 4.1.

4.4.3 Handling Structured Noisy Differences

The algorithm $\mathcal{C}$ above requires knowing the exact differences between the hidden inputs. In this section, we modify $\mathcal{C}$ to allow for using noisy differences $\{\tilde{D}elta^t\}$ with certain *structured* errors. The modified algorithm $\mathcal{C}'$ will be very instrumental later for solving correlated CSP without knowing the differences between hidden inputs.

The noisy differences $\{\tilde{D}elta^t\}_{t\in T}$ that the modified algorithm $\mathcal{C}'$ can work with satisfy that *for every input bit i*, the fraction of errors in the differences for that bit $\{\tilde{D}elta_i^t\}_{t\in T}$ is bounded by a sufficiently small constant γ—that is, the number of $t \in [T]$ s.t. $\tilde{D}elta_i^t \neq \Delta_i^t$ is bounded by γT—in this case, we say that $\{\tilde{D}elta^t\}$ have *γ-bounded errors* (w.r.t. $\{\Delta^t\}$).

The algorithm $\mathcal{C}'$ on input $G, y, y^1, \ldots, y^T$, and noisy differences $\{\tilde{D}elta^t\}$ with *γ-bounded errors*, proceeds identically to $\mathcal{C}$, except in the second step. In the second step, algorithm $\mathcal{C}$ finds the encoding $E_k = \text{Encode}(x_{G(k)})$ for each output bit k, by filling in every entry $E_k[a] = P(x_{G(k)} \oplus a)$ in the encoding with the value y_k^t for *some* t such that $\Delta^t_{G(k)} = a$. However, $\mathcal{C}'$ does not know the actual differences, and following the same procedure would lead to incorrect encodings, which then lead to an inconsistent solution. To overcome this, note that the noisy differences for each input bit $\{\tilde{D}elta_i^t\}$ has bounded errors (by a fraction of γ), and so does $\{\tilde{D}elta^t_{G(k)}\}$ (by a $d\gamma$ fraction). By taking majority of the values y_k^t for all t such that $\tilde{D}elta^t_{G(k)} = a$, the algorithm $\mathcal{C}'$ still derives the correct $E_k[a]$ values with high probability. Therefore, with high probability, $\mathcal{C}'$ recovers every encoding E_k correctly in the second step, and the rest of the algorithm go through identically as $\mathcal{C}$. Below we describe only the modified second step formally.

Modified Step 2: For every output bit k, maintain an array A_k of sets, each initialized as empty and then do: For every $t \in [T]$, add y_k^t to the $(\tilde{D}elta^t_{G(k)})^{\text{th}}$ entry in A_k, that is, $A_k[\tilde{D}elta^t_{G(k)}] = A_k[\tilde{D}elta^t_{G(k)}] \cup \{y_k^t\}$.

If there is an output bit $k \in [m]$, such that, one of the entry in $A_k[a]$ has less than $(\varepsilon^d/2)T$ elements, output $\perp$. Otherwise, for every $k \in [m]$, set the a^{th} entry in its encoding E_k to the majority of the bits in the set $A_k[a]$.

Lemma 4.1. *For every $\varepsilon \in (0, 1/2]$, there is a polynomial time algorithm $\mathcal{C}'$, such that for every predicate $P : \{0,1\}^d \to \{0,1\}$, density $D \geq 1$, and constant $r > 0$, $\mathcal{C}'$ solves worst-case (ε, T)-correlated-CSP(P, D), given noisy differences that have $(\gamma = \varepsilon^d/5d)$-bounded errors, with success probability $1 - n^{-r}$, provided that $T > 8(d + \log(Dn^{r+1}))/\varepsilon^d$.*

Proof. The proof of Theorem 4.1 shows that if for every output bit k, a complete and correct encoding E_k is obtained, solving the linear system derived from these encodings as in Step 3 and 4 of $\mathcal{C}$ yields a consistent solution. Therefore, towards the theorem, it suffices to show that at the end of the modified second step, the probability that any encoding E_k is incomplete or incorrect is bounded by $1/n$.

Consider any output bit k. By the fact that $\{\tilde{Delta}^t\}$ has $(\gamma = \varepsilon^d/5d)$-bounded errors, for each input bit $i \in G(k)$, the fraction of instances s.t. $\tilde{Delta}_i^t \neq \Delta_i^t$ is bounded by $\varepsilon^d/5d$. Therefore, the fraction of instances s.t. $\tilde{Delta}_{G(k)}^t \neq \Delta_{G(k)}^t$ is bounded by $d\gamma = \varepsilon^d/5$. By Claim 4.2, when $T \geq 8(d+\log(mn^r))/\varepsilon^d$, except with probability n^{-r}, for every k and a, the fraction of instances t s.t. $\Delta_{G(k)}^t = a$ is at least $\varepsilon^d/2$. Since the fraction of instances with the wrong differences $\tilde{Delta}_{G(k)}^t \neq \Delta_{G(k)}^t$ is bounded by $\varepsilon^d/5$. Among all instances such that $\tilde{Delta}_{G(k)}^t = a$, more than half of them are correct $\tilde{Delta}_{G(k)}^t = \Delta_{G(k)}^t = a$. Therefore, by taking majority of values y_k^t for all t with $\tilde{Delta}_{G(k)}^t = a$, $\mathcal{C}$ recover the correct value for the a^{th} entry in E_k. Overall, except with probability n^{-r}, every encoding E_k is computed correctly by the modified step 2, which concludes the lemma. □

Remark 4.2. *It is easy to see that Lemma 4.1 can also be extended to invert arbitrary functions with small output-locality with noisy differences: Simply replace the second step in the algorithm $\mathcal{M}$ in Figure 2 with the modified second step above. We omit the details here.*

4.5 Solving Two Correlated CSP Instances with Differences

The algorithm in the last section requires seeing a logarithmic number of auxiliary instances; in this section, we investigate the more stringent scenario where only a single auxiliary instance is available. We present an algorithm $\mathcal{A}$ for solving the average-case $(p^*, 2)$-correlated-CSP(P, D) with differences. However, unlike the algorithm using many auxiliary instances and able to work with generic predicates and parameters, $\mathcal{A}$ only handles a large class $\mathcal{P}$ of predicates, and requires the flipping probability p^* and density D to lie within certain range.

Overview of the Algorithm $\mathcal{A}$ $\mathcal{A}$ works with any predicate $P \in \mathcal{P}$ with the property that one of its first derivative $P^\sigma(a)$ is correlated with one of its input bit a_{i^*}, that is,

$$\exists\ \sigma \in \{0,1\}^d,\ i^* \in [d],\ \gamma > 0, \quad \text{such that} \quad P^\sigma(a) = P(a) \oplus P(a \oplus \sigma) \text{ and } \Pr[P^\sigma(a) = a_{i^*}] \geq 1/2 + \gamma \tag{1}$$

The construction of the algorithm $\mathcal{A}$ additionally depends on the boundary β of P, and the mean ν of the derivative P^σ.

$$\beta = \Pr_{z \xleftarrow{\$} \{0,1\}^d} [\exists z',\ \mathsf{Wt}(z - z') = 1 : P(z) \neq P(z')] \tag{2}$$

$$\nu = \mathbb{E}_{a \xleftarrow{\$} \{0,1\}^d} [P^\sigma(a)] \tag{3}$$

Depending on P^σ, the algorithm $\mathcal{A}$ requires the flipping probability p^* for generating the auxiliary instance to be either sufficiently small if $\sigma_t = 0$, or sufficiently large if $\sigma_t = 1$, and the density D of the CSP instances to be sufficiently large.

$$\exists\ \text{ a sufficiently small } \varepsilon, \text{ such that,} \quad \text{if } \sigma_t = 0,\ p^* = \varepsilon \quad \text{else } \sigma_t = 1,\ p^* = 1 - \varepsilon \tag{4}$$

For any flipping probability p^*, define κ to be the probability that a d-bit string a, where each bit is independently Bernoulli distributed with success probability p^*, matches σ at all but the $i^{*\text{th}}$ bit. That is,

$$\kappa = \Pr[a_1, \cdots, a_d \leftarrow \mathsf{Bernoulli}(p^*)\ :\ a_{\neq t} = \sigma_{\neq t}\ ] \tag{5}$$

Consider an experiment of average-case $(p^*, 2)$-correlated-CSP(P, D) with differences, where a primary instance (G, y) and an auxiliary instance (G, y') are randomly generated as $(G, x, y) \leftarrow \text{rCSP}(P, D)$, (G, y'), $(G, x', y') \leftarrow \text{coCSP}(P, G, x)$, with $\Delta = x \oplus x'$ the difference between the hidden inputs. The algorithm $\mathcal{A}$ on input $((G, y), (G, y'), \Delta)$, tries to find a consistent solution $\tilde{x}$ to the primary instance. At a high level, it proceeds in two steps: First, we design an algorithm $\mathcal{A}_2$ that on input (G, y), (G, y') and Δ, finds an "almost-correct" solution $\tilde{x}$ which is δ-close to a true solution x; then in a second step, we show that such an almost-correct solution can be turned into a true solution by applying the algorithm designed by Bogdanov and Qiao in [BQ12].

Below we formally describe the algorithm $\mathcal{A}$ and prove the following proposition:

Proposition 4.1. *Let P, γ, β, p^* and κ be defined as above, and K a sufficiently large constant.*

$$\varepsilon \leq \frac{\beta^2}{4Kd^6} \qquad \text{and} \qquad D \geq \frac{Kd^8}{\beta^2} + \left(\frac{8}{\gamma\kappa}\right)^2 \ln\left(\frac{6}{\varepsilon}\right).$$

For every $r \geq 1$, there is a polynomial time algorithm $\mathcal{A}$ that solves average-case $(p^*, 2)$-correlated-CSP(P, D) with differences, with probability $1 - O(n^{-r})$.

Note that for any non-trivial P, $\beta \geq 2^{-d}$, for any non-trivial correlation between P^σ and its $i^{*\text{th}}$ input bit, $\gamma > 2^{-d}$, and additionally $\kappa \geq \varepsilon^{-d}$. For a sufficiently large constant N (depending on K), when $\varepsilon \leq 1/N2^{2d}d^6$ and $D \geq N(1/\varepsilon)^{5d}$, the proposition holds. Thus Theorem 4.2 follows directly from this Proposition.

4.5.1 Construction and Analysis of the Algorithm $\mathcal{A}$ (Proof of Theorem 4.2)

To formally describe the algorithm $\mathcal{A}$, we make use of the following notations: For every graph $G : [m] \times [d] \to [n]$, if $G(j, l) = i$, we say that the edge (i, j) has label l. Moreover, we define $\bar{G} : [n] \times [d] \to 2^{[m]}$ to be the "inverse mapping" that maps an input vertex i and a label l to the set of output vertexes that are connected to i with label l; we call them the l-neighbors of i. For brevity of notation, we also denote by $\bar{G}(i)$ the set of all neighbors of input vertex i.

Then, the algorithm $\mathcal{A}$ on input $((G, y), (G, y'), \Delta)$ proceeds in two stages:

Stage 1—Recover an almost-correct solution: For every output bit $k \in [m]$, denote by Evt_k the event where the difference $\Delta_{G(k)}$ restricted to the input bits $x_{G(k)}$ equals to σ; when Evt_k occurs, $\mathcal{A}$ tries to compute derivative P^σ on k's neighbors as follows:

$$z_k = P^\sigma(x_{G(k)}) = P(x_{G(k)}) \oplus P(x_{G(k)} \oplus \sigma) = y_k \oplus y'_k$$

Then, since the derivative P^σ is positively correlated with its $i^{*\text{th}}$ input bit, z_k is positively correlated with $x_{G(k,i^*)}$; thus the value of z_k can be viewed as a vote to the value of $x_{G(k,i^*)}$. Then, for every input bit, taking majority of all votes collected from its i^*-neighbors for which Evt_k holds yields a good estimation of its value. A formal description of the algorithm $\mathcal{A}_2$ in the second stage is provided in Figure 3.

We show that $\mathcal{A}_2$ outputs a string $\tilde{x}$ that matches the hidden input x at all but a small constant fraction of input bits.

Lemma 4.2. *Let P, p^* and κ be defined as above. Fix any constant $\delta_2 \in (0, 1)$. Assume that the density $D \geq (\frac{8}{\gamma\kappa})^2 \ln(\frac{3}{\delta_2})$. $\mathcal{A}_2$ on input $((G, y), (G, y'), \Delta)$ outputs $\tilde{x}$ that is $(\delta_2 + 3\varepsilon/2)$-close to x with overwhelming probability, where variables G, x, y, x', y', Δ are generated in an experiment of average case $(p^*, 2)$-correlated-CSP(P, D) with differences.*

$\mathcal{A}_2$ for finding an almost-correct solution

Let P, D, ε be public parameters. On input (G, y), (G, y'), and difference Δ, proceed in two steps:

1. For every output bit $k \in [m]$, if $\Delta_{G(k)} = \sigma$, compute $z_k = y_k \oplus y'_k$, and vote that $x_{G(k,i^*)}$ has value z_k; otherwise, vote $\perp$ for $x_{G(k,i^*)}$.
2. For every input bit $i \in [n]$, if more than a τ_2 fraction of its i^*-neighbors vote for 1, set $\tilde{x}_i$ to 1, and 0 otherwise. The threshold τ_2 is set to $\kappa\nu$, where κ is defined as in equation (5) and ν is the mean of P^σ as in equation (3).

   Output the estimated string $\tilde{x}$.

Figure 3: Procedure of Stage 1 of the algorithm $\mathcal{A}$

Stage 2—Recover a solution: In this stage, the algorithm tries to turn an almost correct solution $\tilde{x}$ into an fully correct solution. This stage follows the same procedure by Bogdanov and Qiao in [BQ12]. They presented an algorithm for inverting Goldreich's OWF given an almost correct solution. Their algorithm works for any nontrivial predicate P and sufficiently large D. Below we recall a simplified version of their proposition, rephrased in the language of CSP.

Proposition 4.2 (Proposition 4.1 in [BQ12])**.** *Let K be a sufficiently large constant, and β be the boundary of P. Assume that $D \geq Kd^8/\beta^2$ and $\tau \leq \beta^2/(2Kd^6)$. For every $r \geq 1$, there is a polynomial time algorithm $\mathcal{A}_3$ that, for sufficiently large $n \in \mathbb{N}$, on input (G, y) and $\tilde{x}$ that is τ-close to x, outputs $\hat{x}$ consistent with y with probability $1 - O(n^{-r})$, where (G, x, y) is selected at random $(G, x, y) \xleftarrow{\$} \mathrm{rCSP}(P, D)_n$.*

Then, applying the algorithm of Bogdanov and Qiao on (G, y) and the output $\tilde{x}$ of Stage 2 yields $\hat{x}$. By Lemma 4.4 and the above proposition, $\hat{x}$ is consistent with y with probability $1 - O(n^{-r})$.

Concluding Proposition 4.1: Combining Lemma 4.2 and Proposition 4.2, we conclude Proposition 4.1. Set $\delta_2 = \varepsilon/2$. By the parameter setting of ε and D, we have,

$$\varepsilon \leq \frac{\beta^2}{4Kd^6} \qquad \Longrightarrow \qquad \delta_2 + \frac{3\varepsilon}{2} = 2\varepsilon = \tau \leq \frac{\beta^2}{2Kd^6}$$

$$D = \frac{Kd^8}{\beta^2} + \left(\frac{8}{\gamma\kappa}\right)^2 \ln\left(\frac{6}{\varepsilon}\right) \qquad \Longrightarrow \qquad D \geq \frac{Kd^8}{\beta^2} \text{ and } D \geq \left(\frac{8}{\gamma\kappa}\right)^2 \ln\left(\frac{3}{\delta_2}\right)$$

Then, the premises of Lemma 4.2 and Proposition 4.2 are satisfied. Thus, the final output $\hat{x}$ generated in Stage 3 is indeed a true inverse of y with probability $1 - O(n^{-r})$.

Remark 4.3. *The algorithm $\mathcal{A}$ can be used to recover solutions to both the primary and auxiliary instances. This follows since the primary and auxiliary instances (G, y) and (G, y') are symmetric, in the sense that the two instances could have been generated by sampling (G, y') first and then sampling (G, y) as the correlated instance, then the algorithm $\mathcal{A}$ could be applied to recover $\hat{x}'$ consistent with y'. By a union bound, the probability that both $\hat{x}$ and $\hat{x}'$ are correct solutions to y and y' is at least $1 - O(n^{-r})$.*

Now it only remains to prove Lemma 4.2. Towards this, it will be helpful to consider the following randomized procedure for sampling the instances (G, x, y) and (G, x', y') in an experiment

of $(p^*, 2)$-correlated-CSP(P, D), and bound the probabilities that certain corner cases occur. We present the formalization of the randomized procedure in Section 4.5.2 and then prove Lemma 4.2 in Section 4.5.3.

4.5.2 Procedure for Sampling the Primary and Auxiliary Instances

Consider the following procedure for sampling G, x, y, x', y', Δ in an experiment of average-case $(p^*, 2)$-correlated-CSP(P, D) with differences.

1. A difference Δ is sampled by selecting at random each bit Δ_i according to a Bernoulli distribution with success probability p^*. Below, for convenience, we sometimes overload Δ and $\bar{\Delta}$ as the set of input bits with difference 1 and 0 respectively.

   The first corner case—Δ unbalanced: Denote by $\mathsf{Evt}_{\Delta\text{-unbalanced}}(\mu)$, $\mu \in (0, p^*)$, the event that the relative Hamming weight $\mathsf{wt}(\Delta)$ of Δ falls outside the range $[p^* - \mu, p^* + \mu]$. Since each bit of Δ is independently Bernoulli distributed with success probability p^*, by the Chernoff bound, this event occurs with probability at most $\exp(-\Omega(\mu^2 n))$.

2. Select at random sub-strings $x_\Delta \xleftarrow{\$} U_{|\Delta|}$ and $x_{\bar{\Delta}} \xleftarrow{\$} U_{|\bar{\Delta}|}$, and set $x'_\Delta = x_\Delta \oplus 1^{|\Delta|}$ and $x'_{\bar{\Delta}} = x_{\bar{\Delta}}$.

   The second corner case—x unbalanced: Conditioned on the first corner case not occurring, sub-strings x_Δ and $x_{\bar{\Delta}}$ both have length $O(n)$. Then since they are uniformly random, with high probability, their relative Hamming weights $\mathsf{wt}(x_\Delta)$ and $\mathsf{wt}(x_{\bar{\Delta}})$ are centered around $1/2$. Denote by $\mathsf{Evt}_{\text{x-unbalanced}}(\zeta)$ the event that either x_Δ or $x_{\bar{\Delta}}$ is not ζ-balanced, that is, either $\mathsf{wt}(x_\Delta)$ or $\mathsf{wt}(x_{\bar{\Delta}})$ is outside the range $[1/2 - \zeta, 1/2 + \zeta]$.

   Claim 4.3. *Fix any constant $\zeta \in (0, 1/2)$. The probability that event $\mathsf{Evt}_{x\text{-unbalanced}}(\zeta)$ occurs is at most* $\exp(-\Omega(\varepsilon\zeta^2 n))$.

3. Sample graph G by selecting each edge independently as follows: To choose the l^{th} neighbor of output vertex k, $G(k, l)$, first decide whether $G(k, l)$ falls into the set Δ or not at random with probability $\mathsf{wt}(\Delta)$; then, according to the decision, choose a random vertex in Δ or $\bar{\Delta}$ as the neighbor. Finally, compute $y = P \circ G(x)$ and $y' = P \circ G(x')$.

   Independence between choices of neighbors: In the above procedure, the choice of each neighbor $G(k, l)$ and its value $x_{G(k,l)}$ is independent from that of other neighbors $G(k', l')$, $(k', l') \neq (k, l)$. This independence holds even when conditioned on $\mathsf{wt}(\Delta)$, $\mathsf{wt}(x_\Delta)$ and $\mathsf{wt}(x_{\bar{\Delta}})$, and conditioned on these values, the probability that $G(k, l)$ falls into Δ is exactly $\mathsf{wt}(\Delta)$. Furthermore, for any subset of neighbors $\{G(k, l)\}$, conditioned on the decisions that they each belong to Δ or $\bar{\Delta}$, their values $\{x_{G(k,l)}\}$ is 1 with respective probabilities $\mathsf{wt}(x_\Delta)$ or $\mathsf{wt}(x_{\bar{\Delta}})$ *independently.*

   The third corner case—Too few l neighbors: On average, each input vertex has D l-neighbors (i.e., $\mathbb{E}[|\bar{G}(i, l)|] = D$). Since the l^{th} neighbor of each output vertex is chosen independently and randomly from all input vertexes, the set of input vertexes that have too few or too many (comparing to D) l-neighbors are bounded. More precisely, let $S_l \subseteq [n]$ be the set of input bits i that have more than $D/2$ l-neighbors.

   $$S_l = \left\{ i \in [n] \text{ s.t. } |\bar{G}(i, l)| \geq D/2 \right\} \tag{6}$$

   Let $\mathsf{Evt}_{l\text{-nbr}}(\alpha)$ denote the event that S_l contains less than an $1 - \alpha$ fraction of the input vertexes. In [BQ12], Bogdanov and Qiao showed that this event occurs with only exponentially small probability. Below we recall their claim with parameters relevant for our setting.

Claim 4.4 (Lemma A.1 in [BQ12])**.** *Fix any $l \in [d]$ and $\alpha \in (0,1)$. Assume that $D > 32\log(1/\alpha)$. The probability that more than an α fraction of the input vertexes have less than $D/2$ (or more than $2D$ resp.) l-neighbors is* $\exp(-\Omega(\alpha Dn))$.

THE FOURTH CORNER CASE—TOO FEW NEIGHBORS: Finally, we bound the number of neighbors any small constant fraction of input vertexes have. Let $\mathsf{Evt}_{\text{nbrs}}(\beta)$ denote the event that there is a subset $S \subseteq [n]$ of up a β fraction of input vertexes (i.e., $|S| \leq \beta n$) that has more than $2\beta dDn$ neighbors (i.e., $\sum_{i \in S} |\bar{G}(i)| \geq 2\beta dDn$).

Claim 4.5. *Fix any $\beta \in (0,1)$. Assume that $D > 1/(\beta^2 d)$. Then, the probability that event $\mathsf{Evt}_{nbrs}(\beta)$ occurs is at most* $\exp(-(2\beta^2 dD - 1)n)$.

We provide the proofs of Claim 4.3 and 4.5.

Proof of Claim 4.3. As analyzed in the first corner case, except with probability $\exp(-\Omega(\varepsilon^2 n))$, the relative Hamming weight of Δ is in the range $[p^* - \frac{\varepsilon}{2}, p^* + \frac{\varepsilon}{2}]$, that is, $\mathsf{Evt}_{\Delta\text{-unbalanced}}(\mu)$ for $\mu = \varepsilon/2$ does not occur. Therefore, the relative hamming weights of Δ and $\bar{\Delta}$ are at least $\Omega(\varepsilon)$, that is $\mathsf{wt}(\Delta), \mathsf{wt}(\bar{\Delta}) = \Omega(\varepsilon)$. Then, conditioning on this occurring, by the Chernoff bound and the fact that x is chosen at uniformly random, the probability that x_Δ is not ζ-balanced is at most $\exp(-\zeta^2 \mathsf{wt}(\Delta)n) = \exp(-\Omega(\varepsilon\zeta^2 n))$. Similarly, the probability that $x_{\bar{\Delta}}$ is not ζ-balanced is at most $\exp(-\zeta^2 \mathsf{wt}(\bar{\Delta})n) = \exp(-\Omega(\varepsilon\zeta^2 n))$. Then, the claim follows from the union bound. □

Proof of Claim 4.5. We first show that for any particular subset $S \in [n]$ such that $|S| \leq \beta n$, the probability that it has more than $2\beta dDn$ neighbors is exponentially small. In the random graph, each neighbor of each output bit is chosen independently and randomly. Since $|S| \leq \beta n$, the probability that an input bit in S is chosen is at most β. Therefore by the Chernoff bound, the probability that input bits in S are chosen for more than $2\beta dDn$ times is bounded by $\exp(-2\beta^2 dDn)$.

Since the number of subsets of size no bigger than βn is $\binom{n}{\beta n} < 2^n$. by the union bound, the probability that any such subset has more than $2\beta dDn$ neighbors is bounded by $2^n \exp(-2\beta^2 dDn) = \exp(-(2\beta^2 dD - 1)n)$, which is exponentially small when $D \geq 1/\beta^2 d$. □

Next we move to proving Lemma 4.2.

4.5.3 Proof of Lemma 4.2

Recall that the predicate P satisfies that its first derivative P^σ is γ-correlated with the $i^{*\text{th}}$ input bit; ν is the mean of P^σ and κ is the probability that a d-bit string a where each bit is sampled from $\mathsf{Bernoulli}(p^*)$ matches σ at all but the $i^{*\text{th}}$ bit (i.e. $\Pr[a_{\neq t} = \sigma_{\neq i^*}]$).

We prove that $\mathcal{A}_2$ on input $(G, y), (G, y')$ and $\tilde{D}elta = \Delta$ outputs $\tilde{x}$ that is $(\delta_2 + 3\varepsilon/2)$-close to x with overwhelming probability. We prove this statement conditioned on that events $\mathsf{Evt}_{\Delta\text{-unbalanced}}(\mu)$, $\mathsf{Evt}_{\text{x-unbalanced}}(\zeta)$ and $\mathsf{Evt}_{t\text{-nbr}}(\alpha)$ do not occur for $\mu = \min(\gamma\kappa/16d^2, \varepsilon/2)$, $\zeta = \gamma/4d$ and $\alpha = \delta_2/3$. This is without loss of generality as the statement allows for a negligible error probability, and below we analyze $\mathcal{A}_2$ conditioned on $\mathsf{wt}(\Delta) \in [p^* - \mu, p^* + \mu]$, $\mathsf{wt}(x_\Delta) \in [\frac{1}{2} - \zeta, \frac{1}{2} + \zeta]$, $\mathsf{wt}(x_{\bar{\Delta}}) \in [\frac{1}{2} - \zeta, \frac{1}{2} + \zeta]$, and the size of the set of input vertexes with more than $D/2$ i^*-neighbors $|S_{i^*}| \geq (1 - \alpha)n$.

Recall that to estimate the value of x_i, $\mathcal{A}_2$ examines each i^*-neighbor $k \in \bar{G}(i, i^*)$ of i to cast a vote $v_{k,i}$. If it satisfies that the difference $\Delta_{G(k)}$ (restricted to the input bits of the k^{th} output bit) is exactly σ (i.e., $\Delta_{G(k)} = \sigma$)—denote this event by Evt_k—$\mathcal{A}_2$ computes $z_k = y_k \oplus y'_k$, and vote $v_{k,i} = z_k$ as the value of x_i; otherwise, if Evt_k does not occur, it votes $v_{k,i} = \perp$ for x_i. $\mathcal{A}_2$ estimates

$\tilde{x}_i = 1$ if and only if more than a $\tau_2 = \kappa\nu$ fraction of the i^*-neighbors votes for 1. Towards bounding the fraction of input vertexes for which this estimation is wrong, consider separately three different types of input vertexes i:

1. Input bit $i \notin S_{i^*}$: Such input bits do not have enough t-neighbors, and their values can not be recovered using the voting scheme of $\mathcal{A}_2$. However, they only account for at most an $\alpha = \delta_2/3$ fraction of the input bits.

2. Input bit $i \in S_{i^*}$ and $\Delta_i \neq \sigma_{i^*}$: Such input bits have the property that for all its i^*-neighbors, event Evt_k never occurs (since $\Delta_{G(k,i^*)} = \Delta_i \neq \sigma_{i^*}$). Thus the voting scheme of $\mathcal{A}_2$ fails. However, since the probability that $\Delta_i \neq \sigma_{i^*}$ is bounded by $\mathsf{wt}(\Delta) \leq p^* + \mu$ if $\sigma_{i^*} = 0$ and $\mathsf{wt}(\bar{\Delta}) \leq 1 - p^* + \mu$ if $\sigma_{i^*} = 1$. By the way that p^* is set, this probability is bounded by $\varepsilon + \mu \leq 3\varepsilon/2$, since $\mu \leq \varepsilon$.

3. Input bit $i \in S_{i^*}$ and $\Delta_i = \sigma_t$: As discussed above, the fraction of input bits of this type is at least $1 - \delta_2/3 - 3\varepsilon/2$. Thus it suffices to bound the fraction of such input bits for which $\mathcal{A}_2$ gives a wrong estimation. Below, we show that the fraction is bounded by $2\delta_2/3$ with overwhelming probability.

Then, overall, the fraction of input vertexes with wrong estimations is bounded by $\delta_2/3 + 3\varepsilon/2 + 2\delta_2/3 = \delta_2 + 3\varepsilon/2$ with overwhelming probability.

Analyzing Input Bits of the Third Type We show that for every input vertex $i \in S_{i^*}$ and $\Delta_i = \sigma_{i^*}$, the probability that $x_i \neq \tilde{x}_i$ is smaller than $\exp(-\gamma^2\kappa^2 D/32)$. Consider any i^*-neighbor k of i, the probability that k votes 1 to i is

$$\begin{aligned}\Pr[v_{k,i} = 1] &= \Pr[v_{k,i} = 1 \mid \mathsf{Evt}_k] \times \Pr[\mathsf{Evt}_k] \\ &= \Pr[v_{k,i} = 1 \mid \mathsf{Evt}_k] \times \Pr[\Delta_{G(k)} = \sigma]\end{aligned}$$

Given that $\Delta_i = \Delta_{G(k,i^*)} = \sigma_{i^*}$, the second probability is exactly the probability that $\Delta_{G(k)}$ equals to σ at all but the $i^{*\text{th}}$ bit; more precisely, conditioned on $\mathsf{wt}(\Delta)$, this probability equals to

$$\begin{aligned}\Pr[\Delta_{G(k)} = \sigma] &= \Pr[a_1, \cdots, a_d \leftarrow \mathsf{Bernoulli}(\mathsf{wt}(\Delta)) \ : \ a_{\neq i^*)} = \sigma_{\neq i^*}] \\ &\in [\kappa - \mu d^2, \kappa + \mu d^2]\end{aligned}$$

The second line follows from the fact that $\mathsf{wt}(\Delta) \in [p^* - \mu, p^* + \mu]$ and that κ is defined as in equation (5).

On the other hand, when Evt_k occurs, $v_{k,i} = z_k = y_k \oplus y'_k$ equals to the derivative $P^\sigma(x_{G(k)}) = P(x_{G(k)}) \oplus P(x_{G(k)} \oplus \sigma)$. Moreover, conditioned on Evt_k (as well as $\mathsf{wt}(\Delta)$, $\mathsf{wt}(x_\Delta)$, $\mathsf{wt}(x_{\bar{\Delta}})$ and $|S_{i^*}|$), each $x_{G(k,l)}$ with $l \neq i^*$ is independently Bernoulli distributed with probability $\mathsf{wt}(x_\Delta)$ of being 1 if $\sigma_l = 1$ or with probability $\mathsf{wt}(x_{\bar{\Delta}})$ of being 1 if $\sigma = 0$; thus, they are independent and ζ-balanced. Therefore, the value of z_k is $(d-2)\zeta$ close to the derivative $P^\sigma(u_1, \cdots, u_{i^*-1}, x_{G(k,i^*)}, u_{i^*+1}, \cdots, u_d)$ computed on uniformly random $u_1, \cdots, u_{i^*-1}, u_{i^*+1}, \cdots, u_d$. Since P^σ is γ-positively correlated with its $i^{*\text{th}}$ input bit, and has mean ν, we have that,

$$\begin{aligned}&\text{if } x_i = x_{G(k,i^*)} = 1, \qquad \Pr[P^\sigma(u_1, \cdots, u_{i^*-1}, 1, u_{i^*+1}, \cdots, u_d) = 1] = \nu + \gamma/2 \\ &\text{if } x_i = x_{G(k,i^*)} = 0, \qquad \Pr[P^\sigma(u_1, \cdots, u_{i^*-1}, 0, u_{i^*+1}, \cdots, u_d) = 1] = \nu - \gamma/2\end{aligned}$$

Then, since $\zeta = \gamma/4d$,

$$\text{if } x_i = 1, \qquad \Pr[z_k = 1 \mid \mathsf{Evt}_k] \geq \nu + \gamma/2 - (d-2)\zeta = \nu + \gamma/4$$
$$\text{if } x_i = 0, \qquad \Pr[z_k = 1 \mid \mathsf{Evt}_k] \leq \nu - \gamma/2 + (d-2)\zeta = \nu - \gamma/4$$

As $\Pr[v_{k,i} = 1 \mid \mathsf{Evt}_k] = \Pr[z_k = 1 \mid \mathsf{Evt}_k]$, we have that,

$$\text{if } x_i = 1, \qquad \Pr[v_{k,i} = 1] \geq (\nu + \gamma/4)(\kappa - \mu d^2) \geq \nu\kappa + \gamma\kappa/8$$
$$\text{if } x_i = 0, \qquad \Pr[v_{k,i} = 1] \leq (\nu - \gamma/4)(\kappa + \mu d^2) \leq \nu\kappa - \gamma\kappa/8$$

where the last inequalities follow since $\mu \leq \gamma\kappa/16d^2$.

Since the votes from a i^*-neighbor k of i depends only on the choices of k's non-i^*-neighbors and their values, which are independent (when conditioned on $\mathsf{wt}(\Delta)$, $\mathsf{wt}(x_\Delta)$, $\mathsf{wt}(x_{\bar{\Delta}})$ and $|S_{i^*}|$), by the Chernoff bound, when $x_i = 1$ (or $x_i = 0$ resp.), the probability that less than (or more than resp.) a $\tau_2 = \kappa\nu$ fraction of its 2-neighbors vote for 1 is at most $\exp(-\gamma^2\kappa^2 D/64)$. Therefore, the estimation $\tilde{x}_i$ is wrong with probability at most $\exp(-\gamma^2\kappa^2 D/64)$.

Finally, we bound the fraction of i of the third type that has a wrong estimation. Let T be the set of i such that $i \in S_{i^*}$ and $\Delta_i = \sigma_{i^*}$. As discussed above T contains at least a $1-\delta_2/3-3\varepsilon/2$ fraction of the input bits. For every $i \in T$, whether the estimation is wrong $\tilde{x}_i \neq x_i$ depends only on the random choices related to its i^*-neighbors (namely, the choices and values of their non-2-neighbors); since the i^*-neighbors of different input bits are disjoint, the correctness of the estimations is independent. Thus by the Chernoff bound, the probability that more than an $\exp(-\gamma^2\kappa^2 D/64) + \delta_2/3$ fraction of input vertexes of the third type has the estimation wrong is bounded by $\exp(-\Omega(\delta_2^2|T|)) = \exp(-\Omega(\delta_2^2 n))$. When when $D \geq \frac{64}{\gamma^2\kappa^2}\ln(\frac{3}{\delta_2})$, the fraction of wrong estimation is bounded by

$$\exp(-\gamma^2\kappa^2 D/64) + \delta_2/3 \leq 2\delta_2/3,$$

with overwhelming probability. This concludes Lemma 4.2.

4.6 Solving Two Correlated CSP Instances without Differences

In this section, we show that the requirement of knowing the differences between the hidden inputs can be removed, by designing an algorithm $\mathcal{A}_1$ that given only the primary and auxiliary instances $((G, y), (G, y'))$ (generated in an experiment of average-case $(p^*, 2)$-correlated-CSP(P, D)), estimates the differences $\tilde{D}elta$ between the hidden inputs x and x'. Our algorithm $\mathcal{A}_1$ works with any non-trivial predicate P; it additionally requires the flipping probability p^* to be sufficiently small, and the density D to be sufficiently large.

Given such a difference-estimation algorithm $\mathcal{A}_1$, we obtain an algorithm $\mathcal{A}'$ for solving solving average-case $(p^*, 2)$-correlated-CSP(P, D) (without differences), by combining $\mathcal{A}_1$ and $\mathcal{A}$ designed in the last section that requires knowing the differences. However, since $\mathcal{A}'$ requires the flipping probability to be sufficiently small, the combined algorithm $\mathcal{A}'$ only handles predicate P satisfying condition (1) w.r.t. a derivative P^σ and input bit a_{i^*}, such that, $\sigma_{i^*} = 0$. (This is because, if $\sigma_{i^*} = 1$, $\mathcal{A}$ requires the flipping probability $p^* = 1 - \varepsilon$ to be sufficiently large, which contradicts with the requirement by $\mathcal{A}_1$. On the other hand, if $\sigma_{i^*} = 0$, $\mathcal{A}$ requires the flipping probability $p^* = \varepsilon$ to be sufficiently small as $\mathcal{A}_1$ does.) Below, we restrict our attention to such predicates, and use ε directly as the flipping probability. The construction of $\mathcal{A}'$, particularly $\mathcal{A}_1$, additionally depends on the *maximal influence* ρ of different input bits of P, defined as:

$$\ell = \arg\max_{l \in [d]}(\mathrm{Inf}_P(l)) \qquad \rho = \mathrm{Inf}_P(\ell)$$

Below we first formally describe $\mathcal{A}'$ and then prove the following proposition.

Proposition 4.3. *Let P, β, ρ and κ be defined as above. Let M be a sufficiently large constant. Assume that*

$$\varepsilon \le \frac{\rho\beta^2}{Md^6} \qquad \text{and} \qquad D \ge M\left(\frac{d^8}{\beta^2}+\frac{1}{\rho^2}+\frac{1}{\gamma^2\kappa^2}\right)\ln\frac{d}{\gamma\kappa\varepsilon}$$

For every $r \ge 1$, there is a polynomial time algorithm $\mathcal{A}'$ that solves average-case $(\varepsilon, 2)$-correlated-CSP(P, D), with probability $1 - O(n^{-r})$.

Since for any non-trivial predicate, $\rho, \beta \ge 2^{-d}$, for any non-trivial correlation $\gamma \ge 2^{-d}$, and $\kappa \ge \varepsilon^d$, when $\varepsilon \le 1/N2^{3d}d^6$ and $D \ge N/\varepsilon^{5d}$ for a sufficiently large N, the proposition holds. Thus, Theorem 4.3 follows directly from the above proposition.

4.6.1 Construction and Analysis of the Algorithm $\mathcal{A}'$ (Proof of Theorem 4.3)

On input $((G, y), (G, y'))$, $\mathcal{A}'$ proceeds in the following three stages:

Stage 1—Estimate the difference: This stage tries to uncover the difference $\Delta = x \oplus x'$ between x and x'. Intuitively, since the ℓ^{th} input bit of P has high influence ρ, if (the value of) an output bit $k \in [m]$, $m = Dn$, flips in y and y', it indicates that its ℓ^{th} neighbor $G(k, \ell)$ flips in x and x' with some probability. For every input bit i, viewing such an indication as a vote and computing the majority of all votes collected from its ℓ-neighbors $\bar{G}(i, \ell)$ yields a good estimation on whether i flips or not. If input vertex i has sufficiently many ℓ-neighbors, the estimation will be correct with high probability. A formal description of the algorithm $\mathcal{A}_1$ in this stage is provided in Figure 4.

$\mathcal{A}_1$ for estimating the difference Δ

Let P, D, ε be public parameters. On input (G, y) and (G, y'), where $G : [m] \times [d] \to [n]$, $m = Dn$, proceed in two steps:

1. For every output bit $k \in [m]$, if $y_k \ne y'_k$, vote 1 for its ℓ^{th} neighbor $G(k, \ell)$.
2. For every input bit $i \in [n]$, among all its ℓ-neighbors $\bar{G}(i, \ell)$, if more than a $\tau_1 = \rho/5$ fraction of them vote 1, set $\tilde{D}elta = 1$; otherwise, set $\tilde{D}elta = 0$. Output the estimated difference $\tilde{D}elta$.

Figure 4: Procedure of Stage 1 of the algorithm $\mathcal{A}'$

Let $\tilde{D}elta$ be the estimated difference. We show that if ε is sufficiently small and D is sufficiently large, $\tilde{D}elta$ is correct for all but a small constant fraction of bits, with overwhelming probability.

Lemma 4.3. *Let P be any non-trivial predicate and ρ be as defined above. Fix any constant $\delta_1 \in (0, 1)$. Assume that $\varepsilon \le \rho/15d$, and $D \ge \frac{100}{\rho^2}\ln(\frac{3}{\delta_1})$. For every $n \in \mathbb{N}$, $\mathcal{A}_1$ on input $(G, y), (G, y')$ outputs $\tilde{D}elta$ that is δ_1-close to Δ with overwhelming probability, where variables G, x, y, x', y' are generated in an experiment of average-case $(\varepsilon, 2)$-correlated-CSP(P, D), and $\Delta = x \oplus x'$.*

Stage 2—Recover an almost-correct solution: Invoke algorithm $\mathcal{A}_2$ in Section 4.5 with input $(G, y), (G, y'), \tilde{D}elta$, to obtain a solution $\tilde{x}$. We show that even though the estimated difference $\tilde{D}elta$ has some errors, the error rate is so small that it only influences the correctness of a small constant fraction of input bits. Thus, even when receiving a noisy difference $\tilde{D}elta$, $\mathcal{A}_2$ still outputs an almost-correct solution.

Lemma 4.4. *Let P, γ, κ and ρ be defined as above. Fix any constant $\delta_3 \in (0,1)$. Assume that $\varepsilon \le \rho/15d$ and*

$$D \ge \left(\left(\frac{10}{\rho}\right)^2 + \left(\frac{40}{\gamma\kappa}\right)^2\right) \ln \frac{123d}{\gamma\kappa\delta_3}.$$

$\mathcal{A}_2$ on input $((G,y),(G,y'),\tilde{D}elta)$ outputs $\tilde{x}$ that is $(\delta_3+3\varepsilon/2)$-close to x with overwhelming probability, where variables G, x, y, x', y' are generated in an experiment of average case $(\varepsilon,2)$-correlated-CSP(P,D) with differences, and $\tilde{D}elta = \mathcal{A}_1((G,y),(G,y'))$.

Stage 3—Recover a solution: Apply the algorithm of Bogdanov and Qiao [BQ12] on (G,y) and the almost-correct solution $\tilde{x}$ produced in Stage 2, to obtain a true solution $\hat{x}$.

Concluding Proposition 4.3: We conclude Proposition 4.3 by combining Lemma 4.4 and Proposition 4.2. Set $\delta_3 = \varepsilon/2$. We have that for a sufficiently large M.

$$\begin{aligned}
& \varepsilon \le \frac{\rho\beta^2}{Md^6} && \text{and} && D \ge M\left(\frac{d^8}{\beta^2} + \frac{1}{\rho^2} + \frac{1}{\gamma^2\kappa^2}\right)\ln\frac{d}{\gamma\kappa\varepsilon} \\
\implies \quad & \delta_2 + \frac{3\varepsilon}{2} = 2\varepsilon = \tau \le \frac{\beta^2}{2Kd^6} && \text{and} && D \ge \frac{Kd^8}{\beta^2} \\
\implies \quad & \varepsilon \le \rho/15d && \text{and} && D \ge \left(\left(\frac{10}{\rho}\right)^2 + \left(\frac{40}{\gamma\kappa}\right)^2\right)\ln\frac{123d}{\gamma\kappa\delta_3}
\end{aligned}$$

Thus, the premises of Lemma 4.4 and Proposition 4.2 are satisfied. Then, the premises of Lemma 4.2 and Proposition 4.2 are satisfied. Thus, the final output $\hat{x}$ generated in Stage 3 is indeed a true inverse of y with probability $1 - O(n^{-r})$.

Below we provide proofs of Lemma 4.3 and 4.4. Both proofs considers the same randomized procedure for generating the primary and auxiliary instances as described in Section 4.5.2.

4.6.2 Proof of Lemma 4.3

We prove that $\mathcal{A}_1$ on input $(G,y),(G,y')$ outputs $\tilde{D}elta$ that is δ_1-close to Δ with overwhelming probability. We prove this statement conditioned on that events $\mathsf{Evt}_{\Delta\text{-unbalanced}}(\mu)$, $\mathsf{Evt}_{\text{x-unbalanced}}(\zeta)$ and $\mathsf{Evt}_{\ell\text{-nbr}}(\alpha)$ do not occur for $\mu = \varepsilon/2$, $\zeta = \rho/2d$ and $\alpha = \delta_1/3$. This is without loss of generality, since the statement allows for a negligible error probability, and these events only occur with exponentially small probabilities. Below, we analyze $\mathcal{A}_1$ conditioned on the relative Hamming weights $\mathsf{wt}(\Delta)$, $\mathsf{wt}(x_\Delta)$, and $\mathsf{wt}(x_{\bar{\Delta}})$, as well as the number of input vertexes with more than $D/2$ ℓ-neighbors $|S_\ell|$. When the above events do not occur, it holds that $\mathsf{wt}(\Delta) \in [\frac{\varepsilon}{2}, \frac{3\varepsilon}{2}]$, $\mathsf{wt}(x_\Delta) \in [\frac{1}{2}-\zeta, \frac{1}{2}+\zeta]$, $\mathsf{wt}(x_{\bar{\Delta}}) \in [\frac{1}{2}-\zeta, \frac{1}{2}+\zeta]$, and $|S_\ell| \ge (1-\alpha)n$.

Recall that each input vertex i is estimated to be flipped, and $\tilde{D}elta_i$ is set to 1, if and only if more than a $\tau_1 = \rho/5$ fraction of its ℓ neighbors voted 1 for it, where each ℓ neighbor k votes 1 if its value flips $y_k \neq y'_k$. To bound the fraction of input bits i for which this estimation is wrong, we consider separately these that have more than $D/2$ ℓ-neighbors (i.e., $i \in S_\ell$) and these that do not. Since the latter ones only accounts for a $\delta_1/3$ fraction, it suffices to bound the probability that the estimation is wrong $\tilde{D}elta_i \neq \Delta_i$ for these input vertexes in S_ℓ. Towards this, consider the following two cases: The value of input bit i actually flips or not.

Case 1—$i \in S_\ell$ **and** $\Delta_i = 1$**:** We show that for every ℓ-neighbor k of i (i.e., $k \in \bar{G}(i,\ell)$), the probability that k votes for i is at least $\tau_1 + \rho/10$.

$$\begin{aligned}\Pr[k \text{ votes for } i] &= \Pr[y_k \neq y'_k] \\ &\geq \Pr[y_k \neq y'_k \ \wedge\ \forall l \neq \ell,\ x_{G(k,l)} \text{ does not flip}] \\ &= \Pr[y_k \neq y'_k \mid \forall l \neq \ell,\ x_{G(k,l)} \text{ does not flip}]\ \Pr[\forall l \neq \ell,\ x_{G(k,l)} \text{ does not flip}]\end{aligned}$$

The second probability is $(1 - \mathsf{wt}(\Delta))^{d-1} \geq (1 - \frac{3\varepsilon}{2})^{d-1}$. Next, we lower bound the first probability. Conditioned on that a neighbor $G(k,l)$ does not flip, its value $x_{G(k,l)}$ is 1 with probability $\mathsf{wt}(x_{\bar{\Delta}})$; thus, $x_{G(k,l)}$ is ζ-balanced for $\zeta = \rho/2d$. Since the first probability depends only on the values $x_{G(k,l)}$ for $l \neq \ell$, it is $(d-1)\zeta$ close to the same probability when all $x_{G(k,l)}$ were uniformly random, which is at least ρ as $\mathrm{Inf}_P(\ell) \geq \rho$. Therefore,

$$\Pr[y_k \neq y'_k \mid \forall l \neq \ell,\ x_{G(k,l)} \text{ does not flip}] \geq \rho - d\zeta = \frac{\rho}{2}$$

Therefore, we have that

$$\begin{aligned}\Pr[k \text{ votes for } i] &\geq \frac{\rho}{2}\left(1 - \frac{3\varepsilon}{2}\right)^{d-1} \\ &\geq \frac{\rho}{2}\left(1 - (d-1)\frac{3\varepsilon}{2}\right) \geq \frac{3\rho}{10} \geq \tau_1 + \frac{\rho}{10}\end{aligned}$$

where the last inequalities follow from the fact that $\varepsilon < \rho/15d$ and $\tau_1 = \rho/5$.

Case 2—$i \in S_\ell$ **and** $\Delta_i = 0$**:** We show that for every ℓ-neighbor $k \in \bar{G}(i,\ell)$ of i, the probability that it votes for i is at most $\tau_1 - \rho/10$. This follows easily as

$$\begin{aligned}\Pr[k \text{ votes for } i] &= 1 - \Pr[y_k \text{ does not flip}] \\ &\leq 1 - \Pr[\forall l \neq \ell,\ x_{G(k,l)} \text{ does not flip}] \\ &= 1 - (1 - \mathsf{wt}(\Delta))^{d-1} \\ &\leq 1 - (1 - \frac{3\varepsilon}{2})^{d-1} \leq (d-1)\frac{3\varepsilon}{2} \leq \frac{\rho}{10} \leq \tau_1 - \frac{\rho}{10}\end{aligned}$$

where the last inequalities follow again from the fact that $\varepsilon < \rho/15d$ and $\tau_1 = \rho/5$.

Note that the probability that each of ℓ-neighbor k votes for i or not depends only on the choice of k's non-ℓ-neighbors and their values, which are independent (when conditioned on $\mathsf{wt}(\Delta)$, $\mathsf{wt}(x_\Delta)$, $\mathsf{wt}(x_{\bar{\Delta}})$ and $|S_\ell|$). Then by the Chernoff bound, for every input bit $i \in S_\ell$ that has more than $D/2$ ℓ neighbors, the probability that less than a τ_1 fraction of its ℓ-neighbors k votes correctly for i (i.e., k votes 1 for i if $\Delta_i = 1$ and does not vote for i otherwise) is smaller than $\exp(-D\rho^2/100)$. Thus, for every $i \in S_\ell$, the probability that $\Delta_i \neq \tilde{D}elta_i$ is smaller than $\exp(-D\rho^2/100)$.

Furthermore, for every $i \in S_\ell$, whether $\Delta_i = \tilde{D}elta_i$ holds or not depends only on the random choices related to its ℓ-neighbors (namely, the choices of their neighbors and the values of their neighbors). Since the ℓ-neighbors of different input bits i and i' are disjoint, these random choices are independent (when conditioned on $\mathsf{wt}(\Delta)$, $\mathsf{wt}(x_\Delta)$, $\mathsf{wt}(x_{\bar{\Delta}})$ and $|S_\ell|$). Therefore, by the Chernoff bound, the probability that more than a $\exp(-D\rho^2/100) + \delta_1/3$ fraction of $i \in S_\ell$ has $\Delta_i \neq \tilde{D}elta_i$ is no more than $\exp(-\Omega(\delta_1^2|S_\ell|))$, which is in turn bounded by $\exp(-\Omega(\delta_1^2(1-\delta_1/3)n))$ since $|S_\ell| >$

$(1-\delta_1/3)n$. Overall, except with negligible probability, the fraction of i such that $\Delta_i \neq \tilde{D}elta_i$ is at most

$$\begin{aligned} & (\exp(-D\rho^2/100)+\delta_1/3)\tfrac{|S_\ell|}{n} + \tfrac{n-|S_\ell|}{n} \\ \leq \quad & (\delta_1/3+\delta_1/3)(1-\delta_1/3)+\delta_1/3 < \delta_1 \quad [\text{for } D \geq \frac{100\ln(3/\delta_1)}{\rho^2}] \end{aligned}$$

4.6.3 Proof of Lemma 4.4

We show that $\mathcal{A}_2$ on input $(G,y),(G,y')$ and $\tilde{D}elta$ produced by $\mathcal{A}_1$, outputs $\tilde{x}$ that is $(\delta_3+\varepsilon)$-close to x with overwhelming probability. Towards this, as a warm-up, below we first show that by simply combining Lemma 4.2 and 4.3, we can already bound the fraction of errors by a slightly worse bound than $\delta_3+\varepsilon$. After that, we improve the bound to $\delta_3+\varepsilon$ by slightly modifying the analysis of Lemma 4.2. In the analysis below, we set $\delta_1=\delta_2=\delta=\frac{\kappa\gamma}{41d}\delta_3$; thus, we have that,

$$D=\left(\left(\frac{10}{\rho}\right)^2+\left(\frac{40}{\gamma\kappa}\right)^2\right)\ln\frac{123d}{\kappa\gamma\delta_3} \quad\Longrightarrow\quad D\geq\left(\frac{10}{\rho}\right)^2\ln\frac{3}{\delta_1} \text{ and } D\geq\left(\frac{40}{\gamma\kappa}\right)^2\ln\frac{3}{\delta_2}$$

Warm-up: Since $\varepsilon\leq\rho/15d$ and $D\geq\left(\frac{10}{\rho}\right)^2\ln\frac{3}{\delta_1}$, by Lemma 4.3, the estimated difference $\tilde{D}elta$ is δ_1-close to Δ; let E be the set of input vertexes for which the estimated difference is wrong (i.e., $E=\{i \text{ s.t. } \tilde{D}elta_i\neq\Delta_i\}$) and it holds that $|E|\leq\delta_1 n$. Moreover, since $\varepsilon\leq 1/2d$ and $D\geq\left(\frac{8}{\gamma\kappa}\right)^2\ln\frac{3}{\delta_2}$, by Lemma 4.2, if $\mathcal{A}_2$ were given the actual difference Δ, it would output $\tilde{x}$ that is $(\delta_2+3\varepsilon/2)$-close to the actual pre-image x. However, when replacing Δ with the noisy difference $\tilde{D}elta$, the correctness of $\tilde{x}$ may be affected. We argue that this affect is limited. Note that by construction of $\mathcal{A}_2$, the estimated value $\tilde{x}_i$ of the i^{th} input vertex only depends on the choices and values of the neighbors of its i^*-neighbors (i.e., the set $G(\bar{G}(i,i^*))$). If $\tilde{D}elta$ were correct on all relevant 2-hop neighbors, that is, $\tilde{D}elta_{G(\bar{G}(i,i^*))}=\Delta_{G(\bar{G}(i,i^*))}$, the estimated value $\tilde{x}_i$ output by $\mathcal{A}_2$ remains the same no matter it receives $\tilde{D}elta$ or Δ as input.

Next, we bound the number of input vertexes i that has a i^*-neighbor that is connected to E. By Claim 4.5, with overwhelming probability, event $\mathsf{Evt}_{\text{nbrs}}(\delta_1)$ does not occur, that is, every set of input bits of size smaller than or equal to $\delta_1 n$ has at most $2\delta_1 dDn$ neighbors. Since $|E|\leq\delta_1 n$, the set E has at most $2\delta_1 dDn$ neighbors. Therefore, at most $2\delta_1 dDn$ input vertexes would have a i^*-neighbor that is connected to E. As argued above, only the estimation $\tilde{x}_i$ for these input vertexes may be affected when replacing Δ with $\tilde{D}elta$, while the estimation of the rest input vertexes remains the same. As a result, $\mathcal{A}_2$ when receiving $\tilde{D}elta$ as input outputs $\tilde{x}$ that is at most a $(\delta_2+3\varepsilon/2+2\delta_1 dD)$-close to x.

Full Analysis: In the above analysis, for each i, as long as one of its i^*-neighbor is connected with E, the estimated value $\tilde{x}_i$ is counted as wrong. However, $\mathcal{A}_2$ computes $\tilde{x}_i$ by taking majority of votes from all i^*-neighbors of i. Such a voting scheme is much more fault tolerant, in particular, we show that for a large $\delta_2+\varepsilon$ fraction of input vertexes, it holds that as long as the number of i^*-neighbors connected to E is below $\kappa\gamma D/24$, then the estimated value $\tilde{x}_i$ remains the same when replacing Δ with $\tilde{D}elta$. This leads to a tighter bound on the fraction of estimations $\{\tilde{x}_i\}$ whose values (potentially) change when $\tilde{D}elta$ is used.

Towards this, we modify the analysis in the proof of Lemma 4.2 as follows. Recall that the proof of Lemma 4.2 showed the following intermediate statement: When $\mathcal{A}_2$ receives the actual difference

$\tilde{D}elta = \Delta$, for every $i \in T$ of the third type—recall that $T = \{i : i \in S_{i^*} \text{ and } \Delta_i = \sigma_{i^*}\}$, which equals to $(S_{i^*} \cap \bar{\Delta})$ since $\sigma_{i^*} = 0$ for this lemma—the probability that a i^*-neighbors k of i votes 1 is

$$\begin{aligned} &\text{if } x_i = 1, \qquad &&\Pr[v_{k,i} = 1] \geq \nu\kappa + \gamma\kappa/8 \\ &\text{if } x_i = 0, \qquad &&\Pr[v_{k,i} = 1] \leq \nu\kappa - \gamma\kappa/8 \end{aligned}$$

and all votes are independent (when conditioned on $\mathsf{wt}(\Delta) \in [\varepsilon - \mu, \varepsilon + \mu]$, $\mathsf{wt}(x_\Delta) \in [\frac{1}{2} - \zeta, \frac{1}{2} + \zeta]$, $\mathsf{wt}(x_{\bar{\Delta}}) \in [\frac{1}{2} - \zeta, \frac{1}{2} + \zeta]$, and $|S_{i^*}| \geq (1-\alpha)n$, for $\mu = \gamma\kappa/16d^2$, $\zeta = \gamma/4d$ and $\alpha = \delta_2/3$). Then by the Chernoff bound, the probability that the fraction of i^*-neighbors of i that vote 1 is equal to or more than $\gamma\kappa/40$ offset the expectation is bounded by $\exp(-(\frac{\gamma\kappa}{40})^2 D)$; in other words, except with this probability, the fraction of votes of 1 is more than $\nu\kappa + \gamma\kappa/10$ if $x_i = 1$, and less than $\nu\kappa + \gamma\kappa/10$ if $x_i = 0$. In this case, if i has up to $\gamma\kappa D/20$ i^*-neighbors connected to E, when replacing Δ with $\tilde{D}elta$, as argued above at most $\gamma\kappa D/20$ votes, that is, at most a $\gamma\kappa/10$ fraction, of these i^*-neighbors may change. Thus, if $x_i = 1$, the fraction of votes of 1 is still above the threshold $\tau_2 = \nu\kappa$, whereas if $x_i = 0$, the fraction is still below the threshold, leading to the correct estimation $\tilde{x}_i = x_i$. In summary, for every input vertex i of the third type, with probability $1 - \exp(-(\frac{\gamma\kappa}{40})^2 D)$, the estimation $\tilde{x}_i$ remains correct even if up to $\gamma\kappa D/20$ of its 2-neighbors are connected to E.

Next, following syntactically the same argument as in the proof of Lemma 4.2, we lower bound the fraction of input vertexes that can tolerate having up to $\gamma\kappa D/20$ of its i^*-neighbors connected to E. Since the estimation for different $i \in T$ is independent, by the Chernoff bound, at least a $1 - \exp(-(\frac{\gamma\kappa}{40})^2 D) - \delta_2/3$ fraction of $i \in T$ are *tolerant*, with probability $1 - \exp(-\Omega(\delta_2^2|T|)) = 1 - \exp(-\Omega(\delta_2^2 n))$. Since $D \geq (\frac{40}{\gamma\kappa})^2 \ln \frac{3}{\delta_2}$, this fraction is at least $1 - 2\delta_2/3$. Overall, with overwhelming probability, for at least a large fraction p of the input vertexes as described below, the estimation remains correct when replacing Δ with $\tilde{D}elta$, provided that no more than $\gamma\kappa D/20$ of its i^*-neighbors are connected to E, where

$$p = (1 - 2\delta_2/3)\frac{|T|}{n} \geq (1 - \delta_2 - 3\varepsilon/2) \qquad \text{as } |T| \geq (1 - \delta_2/3 - 3\varepsilon/2)n$$

Finally, we bound the fraction of i that have more than $\gamma\kappa D/20$ i^*-neighbors connected to E. Conditioned on $\mathsf{Evt}_{\text{nbrs}}(\delta_1)$ not occurring, the number of vertexes that have a i^*-neighbor connected to E is at most $2\delta_1 dDn$. Thus, at most a fraction $p' = 2\delta_1 dD/(\kappa\gamma D/20) \leq 40\delta_1 d/\kappa\gamma$ of vertexes have up to $\kappa\gamma D/20$ i^*-neighbors connected to E. Therefore, at least a $p - p'$ fraction of the input vertexes are tolerant to and indeed satisfy having at most $\kappa\gamma D/20$ i^*-neighbors connected to E; for these input vertexes, the estimation remains correct when using $\tilde{D}elta$ to replace Δ. Therefore, with overwhelming probability, the fraction of wrong estimation is bounded by $1 - (p - p')$,

$$1 - (p - p') \leq \delta_2 + 3\varepsilon/2 + 40\delta_1 d/\kappa\gamma \leq \delta_3 + 3\varepsilon/2$$

where the last inequality follows as $\delta_3 = \frac{41d}{\kappa\gamma}\delta \geq \frac{40d}{\kappa\gamma}\delta_1 + \delta_2$ with $\delta_1 = \delta_2 = \delta$.

4.7 Solving Many Correlated CSP Instances without Differences

In this section, we revisit our first algorithm $\mathcal{C}$ that solves worst-case (ε, T)-correlated-CSP(P, D) with differences in Section 4.4 and show that provided that ε is sufficiently small, D is sufficiently large, and a sufficiently large logarithmic T number of auxiliary instances are available, then the requirement for knowing the differences can be removed, leading to an algorithm $\mathcal{B}$ for approximating average-case (ε, T)-correlated-CSP(P, D) without differences.

Overview of the Algorithm $\mathcal{B}$: Consider an experiment of average-case (ε, T)-correlated-CSP(P, D), where a primary instance (G, y) with hidden input x, together with T correlated instances $\{G, y^t\}_{t\in[T]}$ with hidden inputs $\{x^t\}_{t\in[T]}$ are generated; let $\left\{\Delta^t\right\}$ be the actual differences between the hidden inputs $\Delta^t = x^t \oplus x$. Given $(G, y, y_1, \cdots, y_T)$, the algorithm $\mathcal{B}$ (similar to algorithm $\mathcal{A}'$ in Section 4.6) proceeds in two stages: In the first stage, $\mathcal{B}$ starts by estimating the differences $\Delta^t = x^t \oplus x$ between x^t and x for every $t \in [T]$. Then in the second stage, it invokes the algorithm $\mathcal{C}$ in Section 4.4 with input $(G, y, y_1, \cdots, y_T)$ and the estimated differences $\{\tilde{D}elta^t\}$ produced in the first stage. The challenge is to show that the noises (or errors) in the estimated differences are small and $\mathcal{C}$ is "robust" to such noises, so that, when invoked with the estimated differences, $\mathcal{C}$ still outputs an *approximate solution* $\tilde{x}$ that satisfies almost all output constraints (i.e., $f(\tilde{x})$ matches y at all but a small constant fraction of locations).

Towards this, recall that as shown already in Section 4.4, the algorithm $\mathcal{C}$ can be modified into another algorithm $\mathcal{C}'$ that tolerates certain structured noises in the differences: In particular, as long as the noisy differences $\{\tilde{D}elta^t\}$ have γ-*bounded errors* (i.e., for every i, the number of instances t such that $\tilde{D}elta_i^t \neq \Delta_i^t$ is bounded by γT) and the number of correlated instances is sufficiently large, then $\mathcal{C}'$ finds a consistent solution $\tilde{x}$ of the primary instance with probability $1-1/n$. (See Lemma 4.1). Ideally, to apply $\mathcal{C}'$, we would like to have an algorithm that can estimate the differences with γ-bounded errors; then, $\mathcal{B}$ can simply invoke $\mathcal{C}'$ with the estimated differences to find a solution. Unfortunately, as we discuss later, there is always a constant fraction of "bad" input bits i, for which the noises in their corresponding estimated differences $\left\{\tilde{D}elta_i^t\right\}$ are not bounded by any constant fraction. Furthermore, recall that the algorithm $\mathcal{C}$ eventually reduces the task of finding a solution to the primary instance to the task of solving a system of linear equations; if the constant fraction of "bad" input bits cannot be identified and excluded, the resulting linear equation system has a constant fraction of error, which becomes computationally intractable to solve. To overcome this, we design a new difference-estimation algorithm $\mathcal{B}_\Delta$ which can identify a small constant fraction of the input bits for which reliable estimation cannot be derived, and for the rest of the input bits it outputs estimation of differences with γ-bounded errors. Then combining $\mathcal{B}_\Delta$ and $\mathcal{C}'$ gives the algorithm $\mathcal{B}$ for approximating average-case (ε, T)-correlated-CSP(P, D).

Fix any predicate P. Let $\rho = \rho(P)$ denote the *maximal influence* of different input bits of P, achieved by the ℓ^{th} input bit as in the last section. Below we first present the new algorithm $\mathcal{B}_\Delta$ for estimating the difference, and then show how to combine it with the worst-case inverting algorithm $\mathcal{C}'$ to obtain the final approximation algorithm $\mathcal{B}$, and prove the following proposition.

Proposition 4.4. *Fix any integer $\alpha^* \in (0, 1]$. Assume that for a sufficiently large constant N,*

$$\varepsilon < \rho/(d(\rho+2)+1/2), \qquad D > \frac{Nd}{\varepsilon^2\alpha^*}\log\left(\frac{2}{\varepsilon}\right), \qquad \text{and } T = T(n) \geq Nd^2\log(Dn)/\varepsilon^{2d}.$$

There is a polynomial time algorithm $\mathcal{B}$ that finds $(1-\alpha^*)$-approximate solutions for average-case (ε, T)-correlated-CSP(P, D) with probability $1-3/n$, for sufficiently large $n \in \mathbb{N}$.

For every non-constant predicate P, $\rho \geq 1/2^d$. Therefore, when $\varepsilon < 1/4d2^d$, the above proposition holds, and thus Theorem 4.4 follows directly from the proposition.

4.7.1 The Sub-Algorithm $\mathcal{B}_\Delta$ for Estimating the Differences

We present and analyze the algorithm $\mathcal{B}_\Delta$ in two steps. In the first step, we identify *certain special properties* for every input bit that are *efficiently checkable*; we present $\mathcal{B}_\Delta$ and show that for every

input bit that satisfies this special property, $\mathcal{B}_\Delta$ estimates the differences on this input bit with γ-bounded errors with $1 - 1/\text{poly}(n)$ probability. In the next step, we show that for almost all pairs of graph G and input x, it holds that all but a small constant fraction of the input bits satisfy these special properties, with probability $1 - 1/\text{poly}(n)$. Then combining the two steps, we derive that the algorithm $\mathcal{B}_\Delta$ outputs $\perp$ for every input bit that does not satisfy the special properties (by checking them efficiently), and outputs estimated differences with γ-bounded errors for every input bit that satisfy the special properties, with $1 - 1/\text{poly}(n)$ probability.

Consider any $\varepsilon \in (0, 1/2)$, any hidden input $x \in \{0,1\}^n$, and any bipartite graph $G : [n] \times [d] \to [m]$ for $m = Dn$. Recall that the strings $\{x^t\}$ are derived by randomly and independently flipping each bit of x with probability ε; it is equivalent to first sample the differences $\{\Delta^t\}$ by sampling every bit Δ_i^t from a Bernoulli distribution with success probability ε independently, and then compute $x^t = x \oplus \Delta^t$. When the graph G and input x is fixed, the only randomness comes from the sampling of $\{\Delta^t\}$. Below we define several properties of an input node.

1. We say that an input node i is admissible if it has at least $D/2$ and at most $2D$ ℓ-neighbors, that is, $D/2 \le |\bar{G}(i, \ell)| \le 2D$; let S^* denote this set of input bits.

2. For every input bit $i \in [n]$ and instance $t \in [T]$, let $F(i,t) \in [0,1]$ be the random variable representing the fraction of output bits in i's ℓ-neighbors that have their values flip from y to y^t, that is, the fraction of $k \in \bar{G}(i, \ell)$, such that $y_k \neq y_k^t$.

3. Furthermore, let $\mathbb{E}(i, 1)$ be the expectation of $F(i,t)$ conditioned on that the i^{th} input bit flips (i.e., $\Delta_i^t = 1$) and similarly $\mathbb{E}(i, 0)$ be the expectation of the fraction of flips conditioned on that the i^{th} input bit does not flip (i.e., $\Delta_i^t = 0$).

$$\mathbb{E}(i, b) = \mathbb{E}[F(i,t) \mid \Delta_i^t = b]$$

   Note that these two expectations only depend on the graph G and input x.

Intuitively, if for an input node i, the expected fraction of flips $\mathbb{E}(i, 1)$ conditioned on i flipping is bigger than the expected fraction $\mathbb{E}(i, 0)$ conditioned on i not flipping with a reasonable margin, then there is a way of estimating the differences on that input bit across different instances. Roughly speaking, we show later that when i is admissible, it holds with $(1 - 1/\text{poly}(n))$ probability that the actual numbers of flips $F(i,t)$ in *most* instances t lie in a small region $R_1 = [\mathbb{E}(i,1) - \mu, \mathbb{E}(i,1) + \mu]$ centered around $E(i, 1)$ if i flips in that instance, and in region $R_0 = [\mathbb{E}(i,0) - \mu, \mathbb{E}(i,0) + \mu]$ otherwise. When $\mathbb{E}(i, 1)$ is sufficiently larger than $\mathbb{E}(i, 0)$, R_0 and R_1 are disjoint. If one can approximate the average of the expectations $\tau_i = (\mathbb{E}(i,0) + \mathbb{E}(i,1))/2$, then τ_i can be used as a threshold for estimating whether i flips or not in an instance: In particular, every instance where the actual fraction of flips is above τ_i is categorized as one in which i flips, and every instance where the fraction of flip is below τ_i as one in which i does not flip.

To approximate the average of the expectations, choose an appropriately small constant μ. Partition the range $[0, 1]$ into small regions of size of 2μ—regions $\{\mathcal{B}_q = [2q\mu, 2(q+1)\mu)\}$ for $0 \le q < 1/2\mu$; we call them buckets. We say that a instance t falls into a bucket, if the fraction of flips $F(i,t)$ in that instance lies in that bucket; and we say that a bucket is *heavy* if there are more than $(\varepsilon/2)T$ instances in that bucket. When $\mathbb{E}(i,1) > \mathbb{E}(i,0) + 8\mu$, the two regions R_0 and R_1 are far apart, and with high probability, R_0 contains most instances in which i does not flip (close to $1 - \varepsilon$ fraction), and R_1 contains most instances in which i does not flip (close to ε fraction). Since R_0 and R_1 are both of size 2μ, we expect the following event to hold.

Event $\mathsf{Evt}_{\mathbf{bckt}}(i,\mu)$: There are at most four heavy buckets forming two consecutive pairs (B_{q-1}, B_q) and $(B_{q'}, B_{q'+1})$; they are at least one bucket apart, that is, $q' > q$, and $(B_{q'}, B_{q'+1})$ contains fewer instances than (B_q, B_{q+1}).

(Jumping ahead, we will show later that (B_{q-1}, B_q) intersects with R_0, and $(B_{q'}, B_{q'+1})$ intersecting with R_1 with high probability). It is easy to see that event $\mathsf{Evt}_{\mathrm{bckt}}(i,\mu)$ is efficiently checkable. Furthermore, when the event occurs, we can approximate the average of $E(i,0)$ and $E(i,1)$, using the middle point in between (B_{q-1}, B_q) and $(B_{q'}, B_{q'+1})$, that is, set $\tau_i = (q+q')\mu$.

We formally present the algorithm $\mathcal{B}_\Delta$ in Figure 5.

Algorithm $\mathcal{B}_\Delta$ for estimating the differences between $\{x_t\}$ and x

Let P, D, ε be public parameters. On input $G, y, y^1, y^2, \ldots, y^T$, and μ proceed as follows: The algorithm maintains an array A of estimated differences for every input bit i.

1. Let $S^* \subseteq [n]$ be the set of admissible input bits i, that is, all i that has at least $D/2$ and at most $2D$ ℓ-neighbors, or $S^* = \{i \in [n] \text{ s.t. } D/2 \le |\bar{G}(i,\ell)| \le 2D\}$. For every input node $i \notin S^*$; set the estimated differences $A[i]$ for input bit i to $\perp$.
2. For every input node $i \in S^*$, do:

   For every instance $t \in [T]$, count the fraction of flips $F(i,t)$ in the ℓ-neighbors of i, that is, the fraction of $k \in \bar{G}(i,\ell)$ s.t. $y_k \neq y_k^t$.

   For each bucket $B_q = [2q\mu, (2q+2)\mu)$ that intersects with $[0,1]$, count the number of instances t that fall into that bucket, that is, the number of $t \in [T]$ s.t. $F(i,t) \in B_q$. If event $\mathsf{Evt}_{\mathrm{bckt}}(i,\mu)$ occurs w.r.t. buckets $B_{q-1}, B_q, B_{q'}, B_{q'+1}$, set $\tau_i = (q+q')\mu$; otherwise, set the estimated differences $A[i]$ for input bit i to $\perp$.
3. For every input node $i \in S^*$ for which event $\mathsf{Evt}_{\mathrm{bckt}}(i,\mu)$ occurs, set the estimated differences $A[i]$ to $\{\tilde{D}elta_i^t\}$ as follows: $\tilde{D}elta_i^t = 1$ if $F(i,t) \ge \tau_i$, and $\tilde{D}elta_i^t = 0$ if $F(i,t) < \tau_i$.
4. Finally, output array A.

Figure 5: The algorithm $\mathcal{B}_\Delta$ estimates the differences between the hidden solution x of y and that $\{x_t\}$ of the correlated instances $\{y_x\}$.

Analysis of $\mathcal{B}_\Delta$: Let P, ρ, ℓ be defined as above. Consider an experiment of average-case (ε, T)-correlated-CSP(P, D), where variables $G, x, y, x^1, y^1, \cdots, x^T, y^T$ are generated and $\Delta^t = x^t \oplus x$ for all $t \in [T]$. In the analysis below, we set the following parameters as

$$\eta = \eta(\varepsilon) = 1 - (1-\varepsilon)^{d-1}, \qquad \mu = (\rho - (\rho+2)\eta)/16, \qquad \gamma < \min((1-2\varepsilon)/4, \varepsilon/4)$$

We show that $\mathcal{B}_\Delta$ satisfies the following property:

Lemma 4.5. *Fix any $\alpha \in (0,1)$. Assume that ε is sufficiently small such that $\eta \le \rho/(\rho+2)$, $D > 4\log(4/\gamma)/\mu^2 + 12$ and $T = T(n) > 5\log n/\gamma^2$. For every sufficiently large $n \in \mathbb{N}$, the algorithm $\mathcal{B}_\Delta$ on input μ and $(G, y, y^1, \cdots, y^T)$, outputs an array A satisfying the following two conditions with probability $1 - 1/n$.*

Condition 1: For every admissible input bit i for which $\mathsf{Evt}_{\mathrm{bckt}}(i,\mu)$ occurs, *$A[i]$ equals to a set of estimated differences $\{\tilde{D}elta_i^t\}_{t\in[T]}$ with γ-bound errors w.r.t. the actual differences $\{\Delta_i^t\}_{t\in[T]}$ for input i.*

Condition 2: For the rest input bits, $A[i] = \perp$, *and there are at most a* α *fraction of such input bits.*

Towards proving the lemma, we first prove two instrumental claims. The first claim shows that over the random choices of the graph G, for every $\mathsf{admissible}$ i, except with probability $O(1/n^2)$, most of its ℓ-neighbors do not have any repetitive neighbors, and do not share any neighbor other than i.

Claim 4.6. *For every* $n \in \mathbb{N}$*, over the random choice of the graph* G*, the probability that there is an input bit* $i \in S^*$ *with the following property is* $O(1/n^2)$*: Input bit* i *has more than* 6 ℓ*-neighbors* k*, such that, either* k *has duplicate neighbors (i.e.,* $\exists l \neq l' \in [d]$ *s.t.* $G(k,l) = G(k,l')$*), or it shares a neighbor different from* i *with another* ℓ*-neighbor of* i *(i.e.,* $\exists k' \neq k \in \bar{G}(i,\ell)$*,* $l, l' \in [d]$ *s.t.* $G(k,l) = G(k',l') \neq i$*).*

Proof. For every $i \in S^*$, we show that the probability that the above event occurs w.r.t. i is bounded by $O(1/n^3)$: Conditioned on the set of ℓ-neighbors $k \in \bar{G}(i,\ell)$ of i, the choices of their neighbors that are not labelled as ℓ are uniformly random over $[n]$ and independent. Consider an experiment where these non-ℓ-neighbors are chosen one by one in sequence, if the above event occurs, there are at least 3 times, a newly selected neighbor coincides with one of the previously selected neighbors or i, which occurs with probability at most $2Dd/n$. Therefore, the probability that this occurs 3 times is bounded by $(2Dd/n)^3 = O(1/n^3)$.

Therefore by a union bound, the probability that there is an $i \in S^*$ such that the above event occurs is $O(1/n^2)$. □

The second claim shows that for every input bit $i \in [n]$, the following events occur with $(1 - 1/\mathrm{poly}(n))$ probability.

- *Event* $\mathsf{Evt}_{\textit{xi-flip}}(i,\gamma)$*:* The fraction of instances $t \in [T]$ in which x_i flips (i.e., $\Delta_i^t = 1$) lies in $[\varepsilon - \gamma, \varepsilon + \gamma]$. Correspondingly, the fraction of instances $t \in [T]$ in which x_i does not flip (i.e., $\Delta_i^t = 0$) lies in $[1 - \varepsilon - \gamma, 1 - \varepsilon + \gamma]$.
- *Event* $\mathsf{Evt}_{\textit{nbr-flip}}(i,\gamma,\mu)$*:* For all but a γ fraction of the instances $t \in [T]$, it holds that if x_i flips in the t^{th} instance, the fraction of flips $F(i,t)$ lies in R_1, and if x_i does not flip, the fraction of flips $F(i,t)$ lies in R_0.

Claim 4.7. *Let parameters* ε*,* n*, and* P *be defined as above. Fix any* $\mu, \gamma > 0$*. Assume that* $D > 4\log(4/\gamma)/\mu^2 + 12$*, and* $T > 5\log n/\gamma^2$*. For every* x *and a randomly chosen graph* G*, it holds that the probability that events* $\mathsf{Evt}_{\textit{xi-flip}}(i,\gamma)$ *and* $\mathsf{Evt}_{\textit{nbr-flip}}(i,\gamma,\mu)$ *occur for every* $\mathsf{admissible}$ $i \in [n]$ *is at least* $1 - 1/3n$*, over the random choices of* G *and* $\{\Delta^t\}_{t\in[T]}$*.*

Proof. Since in every instance t, every input bit x_i flips with probability ε independently, by Hoeffding's inequality, the probability that the fraction of instances in which x_i flips lies in $[\varepsilon - \gamma, \varepsilon + \gamma]$ is at least $1 - 3\exp(-2T\gamma^2)$. When $T > 5\log n/\gamma^2$, this probability is at least $1 - 1/6n^2$.

To lower bound the probability that $\mathsf{Evt}_{\text{nbr-flip}}(i,\gamma,\mu)$ occurs, we first show that for every instance $t \in [T]$, the probability that $F(i,t) \notin R_b$ where $b = \Delta_i^t$ is at most $\gamma/2$. For every $k \in \bar{G}(i,\ell)$, let random variable V_k^t denote whether output bit k flips or not in the t^{th} instance. Then, the fraction of flips in $\bar{G}(i,\ell)$ is the average of these random variables.

$$F(i,t) = \frac{1}{|\bar{G}(i,\ell)|}\left(\Sigma_{k\in\bar{G}(i,\ell)} V_k^t\right)$$

Therefore conditioned on $\Delta_i^t = b$,

$$E(i,b) = \mathbb{E}\left[F(i,t) \mid \Delta_i^t = b\right] = \frac{1}{|\bar{G}(i,\ell)|}\left(\Sigma_{k\in\bar{G}(i,\ell)} \Pr[V_k^t = 1 \mid \Delta_i^t = b]\right)$$

let $\bar{G}^*(i,\ell) \subseteq \bar{G}(i,\ell)$ be a set that contains the maximal number of the ℓ-neighbors of i satisfying that the only neighbor they share with each other is i, and none of them have repetitive neighbors, that is, for every $k \in \bar{G}^*(i,\ell)$ and $l \neq l' \in [d]$, it holds that $G(k,l) \neq G(k,l')$ and for every $k \neq k' \in \bar{G}^*(i,\ell)$ it holds that $G(k) \cap G(k') = \{i\}$. We call $\bar{G}^*(i,\ell)$ the set of *non-overlapping* ℓ-neighbors of i. By Claim 4.6, except with probability $O(1/n^2)$, for every input node $i \in S^*$, the number of the non-overlapping ℓ-neighbors of i is at least the number of ℓ-neighbors of i minus 6. Condition on such a graph G; since $i \in S^*$ has at least $D/2$ ℓ-neighbors, the quantities $F(i,t)$ and $E(i,b)$ are at most $(12/D)$-far from their counterparts that are restricted to the set of non-overlapping ℓ-neighbors.

$$\begin{aligned}
\tilde{F}(i,t) &= \frac{1}{|\bar{G}^*(i,\ell)|}\left(\Sigma_{k\in\bar{G}^*(i,\ell)} V_k^t\right) \\
\tilde{E}(i,b) &= \frac{1}{|\bar{G}^*(i,\ell)|}\left(\Sigma_{k\in\bar{G}^*(i,\ell)} \Pr[V_k^t = 1 \mid \Delta_i^t = b]\right)
\end{aligned}$$

Then, further conditioning on $\Delta_i^t = b$ (besides G and x), the value of each random variable V_k^t only depends on the value of Δ^t on the neighbors of k other than i. Since every k in $\bar{G}^*(i,\ell)$ do not share any neighbors other than i, we have that $\{V_k^t\}$ are all independent. Therefore by Hoeffding's inequality and the fact that $|\bar{G}^*(i,\ell)| \geq D/2 - 6$, we have that the average $\tilde{F}(i,t)$ of these independent random variables is $\mu/2$-close to its mean $\tilde{E}(i,b)$, with probability at least $1 - 2\exp(-(D/2-6)\mu^2/2)$. When $D > 4\log(4/\gamma)/\mu^2 + 12$, this probability is greater than $1 - \gamma/2$.

Since the quantities $F(i,t)$ and $E(i,b)$ are $12/D$-close to $\tilde{F}(i,t)$ and $\tilde{E}(i,b)$, we have that for every $i \in S^*$ and every t, conditioned on $\Delta_i^t = b$, the actual fraction of flips $F(i,t)$ is $(\mu/2 + 24/D)$-close to its mean $E(i,b)$, with probability $1 - \gamma/2$; when $\mu < 1/6$ and $D > 4\log(4/\gamma)/\mu^2 + 12$, this implies that $F(i,t)$ is μ-close to its mean $E(i,b)$, or equivalently, $F(i,t) \in R_b$, with probability $1 - \gamma/2$.

Moreover, the random variables $F(i,t)$ depends only on the value Δ^t, which is independent for different instances, the event whether $F(i,t)$ falls outside R_b for $b = \Delta_i^t$ is independent for different t. Thus by Hoeffding's inequality again, the fraction of instances $t \in [T]$ such that this even occurs is at most γ with probability $1 - \exp(-T\gamma^2/2)$. When $T > 5\log n/\gamma^2$, this probability is at least $1 - 1/6n^2$.

Finally, by a union bound the probability that events $\mathsf{Evt}_{\text{xi-flip}}(i,\gamma)$ and $\mathsf{Evt}_{\text{nbr-flip}}(i,\gamma,\mu)$ occur is at least $1 - 1/3n^2$. Then, by a union bound over all $\mathsf{admissible}$ $i \in [n]$, the probability that these two events occur for all $\mathsf{admissible}$ input is at $1 - 1/3n$. □

Now we are ready to prove Lemma 4.5.

Proof of Lemma 4.5. Toward proving the lemma, we analyze the two conditions below:

Analysis of Condition 1: By Claim 4.7, when $D > 4\log(4/\gamma)/\mu^2 + 12$ and $T > 5\log n/\gamma^2$, events $\mathsf{Evt}_{\text{xi-flip}}(i,\gamma)$ and $\mathsf{Evt}_{\text{nbr-flip}}(i,\gamma,\mu)$ occur for all $\mathsf{admissible}$ $i \in [n]$ with probability $1 - 1/3n$. Therefore, it suffices to prove that conditioned on this holds, $\mathcal{B}_\Delta$ outputs estimated differences $\{\tilde{Delta}_i^t\}$ with γ-bounded errors for every $\mathsf{admissible}$ i with $\mathsf{Evt}_{\text{bckt}}(i,\mu)$ occurring.

Fix one such i, the fact that events $\mathsf{Evt}_{\text{xi-flip}}(i,\gamma)$ and $\mathsf{Evt}_{\text{nbr-flip}}(i,\gamma,\mu)$ hold means,

1. For at most a γ fraction of $t \in T$, $F(i,t)$ is outside R_b for $b = \Delta_i^t$; let Q denote this set of t.

2. For instances $t \notin Q$ and $\Delta_i^t = 1$, $F(i,t) \in R_1$. By the fact that at least $\varepsilon - \gamma$ fraction of t has $\Delta_i^t = 1$, R_1 contains at least $\varepsilon - 2\gamma > \varepsilon/2$ fraction of the instances, when $\gamma < \varepsilon/4$.

3. For instances $t \notin Q$ and $\Delta_i^t = 0$, $F(i,t) \in R_0$. By the fact that at least $1 - \varepsilon - \gamma$ fraction of t has $\Delta_i^t = 0$, R_0 contains at least $1 - \varepsilon - 2\gamma > \varepsilon/2$ fraction of the instances, when $\gamma < \varepsilon/4$.

Therefore, only buckets $B_q = [2q\mu, (2q+2)\mu)$ that intersects with R_0 or R_1 may be heavy, whereas all other buckets contains at most a $\gamma < \varepsilon/4$ fraction of instances. Let the four buckets be (B_{q-1}, B_q) and $(B_{q'}, B_{q'+1})$. When $\mathsf{Evt}_{\text{bckt}}(i,\mu)$ occurs, the two pairs are at least one bucket apart, that is, $q' > q$, and $(B_{q'}, B_{q'+1})$ contains more instances than (B_{q-1}, B_q). Since the pair that intersects with R_0 contains at most $\varepsilon + 2\gamma$ fraction of the instances, whereas the pair that interacts with R_1 contains at least $1 - \varepsilon - 2\gamma$ fraction of the instances, when $\gamma > (1-2\varepsilon)/4$, this means $(B_{q'}, B_{q'+1})$ contains R_1 and (B_{q-1}, B_q) contains R_0. Therefore by setting the threshold τ_i to the middle point of these two pairs, every instance in the second and third cases above is categorized correctly, that is $\tilde{Delta}_i^t = \Delta_i^t$. Only instances in the first case above may have an invalid estimation. Thus, the fraction of errors in $\{\tilde{Delta}_i^t\}$ is bounded by γ. This establishes that with probability at least $1 - 1/3n$, condition 1 holds.

Analysis of Condition 2: We show that over the *random choices* of graph G and input x, it holds with $1 - 1/n^2$ probability, the fraction of input bits that is not $\mathsf{admissible}$ or w.r.t. which $\mathsf{Evt}_{\text{bckt}}(i,\mu)$ does not occur is bounded by a small constant. We first bind the probabilities of the following events:

Event 1: The randomly sampled $x \xleftarrow{\$} \{0,1\}^n$ is not $(\rho/2d)$-balanced. By Chernoff bound, the probability that this event occurs is bounded by $\exp(-\Omega(n\rho^2/d^2))$.

Event 2: Less than an $1 - \alpha$ fraction of the input bits are $\mathsf{admissible}$, that is $|S^*| < (1-\alpha)n$. In other words, more than α fraction of the input bits have less than $D/2$ or more than $2D$ ℓ-neighbors. It follows from the same proof as in Claim 4.4 that this occurs with probability $\exp(-\Omega(\alpha D n))$.

Since the above two events occur with exponentially small probability, it suffices to prove the lemma conditioned on that they do not occur, as we do below.

Next, we bound the fraction of input nodes for which $\mathsf{Evt}_{\text{bckt}}(i,\mu)$ does not occur.

By Claim 4.7, (when $D > 4\log(4/\gamma)/\mu^2 + 12$, and $T > 5\log n/\gamma^2$,) it holds that except with probability $1/3n$, for every $i \in S^*$, $\mathsf{Evt}_{\text{xi-flip}}(i,\gamma)$ and $\mathsf{Evt}_{\text{nbr-flip}}(i,\gamma,\mu)$ occur. We show that conditioned on this happening, if $E(i,1) \geq E(i,0) + 8\mu$, event $\mathsf{Evt}_{\text{bckt}}(i,\mu)$ must happen. It follows from the same argument as in the analysis for condition 1 that conditioned on $\mathsf{Evt}_{\text{xi-flip}}(i,\gamma)$ and $\mathsf{Evt}_{\text{nbr-flip}}(i,\gamma,\mu)$ occurring, (when $\gamma < \min((1-2\varepsilon)/4, \varepsilon/4)$,) there are at most four heavy buckets—(B_{q-1}, B_q) intersecting with R_0 and $(B_{q'}, B_{q'+1})$ intersecting with R_1—and if they are disjoint, the latter pair contains fewer instances than the former. Thus to show that $\mathsf{Evt}_{\text{bckt}}(i,\mu)$ occurs, it only remains to show that the two pairs of buckets are at least one bucket apart. This is true when $E(i,1) \geq E(i,0) + 8\mu$: $B_{q'}$ intersects with R_1, and thus $2q'\mu \geq E(i,1) - 3\mu$; B_q intersects with R_0 and thus $(2q+2)\mu \leq E(i,0) + 3\mu$, which shows B_q and B_q are at least one bucket (of size 2μ) away.

Therefore, to lower bound the fraction of input bits for which $\mathsf{Evt}_{\text{bckt}}(i,\mu)$ does not occur, it suffices to lower bound the fraction of input bits for which $E(i,1) \geq E(i,0) + 8\mu$ holds. We show the following claim:

Claim 4.8. *Let ρ be defined as above and $\eta = 1-(1-\varepsilon)^{d-1}$. Assume that ε is sufficiently small such that $\eta < \rho/(\rho+2)$ and $\mu < (\rho-(\rho+2)\eta)/16$. It holds that conditioned on Event 1, every input bits $i \in S^*$ has $E(i,1) \geq E(i,0)+8\mu$, with probability $1-O(1/n^2)$.*

Proof. For every i and b, recall that $E(i,b) = \mathbb{E}[F(i,t)|\Delta_i^t = b]$ for any $t \in [T]$. In other words, $E(i,1)$ (or $E(i,0)$ respectively) is the expected fraction of flips in i's ℓ-neighbors $\bar{G}(i,\ell)$ when each bit in x is flipped with probability ε, conditioned on i flips (or i does not flip respectively). Then,

$$\begin{aligned} E(i,1) &= \frac{1}{|\bar{G}(i,\ell)|}\Sigma_{k\in\bar{G}(i,\ell)}\left(\Pr[k \text{ flips} \mid i \text{ flips}]\right) \\ E(i,0) &= \frac{1}{|\bar{G}(i,\ell)|}\Sigma_{k\in\bar{G}(i,\ell)}\left(\Pr[k \text{ flips} \mid i \text{ does not flips}]\right) \end{aligned}$$

Next we bound the distance between $E(i,1)$ and $E(i,0)$ for every $i \in [n]$, conditioned on Event 1 occurring. We say that an input bit i is influential to its ℓ neighbor k w.r.t. a graph G and an input x, if the value of y_k flips, when x_i flips and all other neighbors of k remain the same, that is, $P(x_{G(k)}) \oplus P((x_{G(k)})^{\oplus\ell}) = 1$. We then have that:

$$\begin{aligned} & \Pr_{\Delta}[k \text{ flips} \mid i \text{ flips} \wedge i \text{ influential to } k] \\ \geq\ & \Pr_{\Delta}[\forall l \neq \ell,\ x_{G(k,l)} \text{ does not flip} \mid i \text{ flips} \wedge i \text{ influential to } k] \\ \geq\ & (1-\varepsilon)^{d-1} = 1-\eta \end{aligned}$$

On the other hand, when the graph G and x is chosen at random, for every $i \in [n]$ and every $k \in \bar{G}(i,\ell)$,

$$\Pr_{G,x}[i \text{ influential to } k] = \Pr_{G,x}[P(x_{G(k)}) \oplus P((x_{G(k)})^{\oplus\ell}) = 1]$$

Conditioned on Event 1 occurring, that is, x is $(\rho/2d)$-balanced, and the fact that every non-ℓ-neighbor of k is chosen at random over $[n]$ independently, we have that for every $l \neq \ell \in [d]$, $x_{G(k,l)}$ is $\rho/2d$ balanced. Thus, the right hand side probability is at most $\rho/2$-far from the probability that $P(z) \oplus P(z^{\oplus\ell}) = 1$ for a uniformly chosen z, which equals to $\mathrm{Inf}_P(\ell) = \rho$. Therefore, we have that the

$$\Pr_{G,x}[i \text{ influential to } k \mid \text{Event 1}] \geq \mathrm{Inf}_P(\ell) - \rho/2 = \rho/2$$

Therefore,

$$\begin{aligned} & \Pr[k \text{ flips} \mid i \text{ flips} \wedge \text{ Event 1}] \\ \geq\ & \Pr_{\Delta}[k \text{ flips} \mid i \text{ flips} \wedge i \text{ influential to } k] \times \Pr_{G,x}[i \text{ influential to } k \mid \text{Event 1}] \\ \geq\ & (1-\eta)\rho/2 \end{aligned}$$

On the other hand,

$$\begin{aligned} & \Pr[k \text{ flips} \mid i \text{ does not flips}] \\ \leq\ & 1 - \Pr[\forall l \neq \ell,\ x_{G(k,l)} \text{ does not flip} \mid i \text{ does not flips}] \\ \leq\ & 1-(1-\varepsilon)^{d-1} = \eta \end{aligned}$$

Since the above inequalities hold for all i and ℓ-neighbor k of i when Event 1 occurs, we have that $E(i,1) \geq (1-\eta)\rho/2$ and $E(i,0) \leq \eta$, and when ε is sufficiently small such that $\eta < \rho/(\rho+2)$ and $\mu \leq (\rho-(\rho+2)\eta)/16$, it indeed holds that $E(i,1) \geq E(i,0)+8\mu$, when Event 1 occurs. This concludes the claim. □

Given the above claim, by a union bound, for every sufficiently large $n \in \mathbb{N}$, except with probability $1/2n$, Event 1, 2 and Claim 4.7 and 4.8 hold, in which case all but an α fraction of the input nodes are admissible and has $\mathsf{Evt}_{\text{bckt}}(i, \ell)$ occurring.

Concluding the lemma: Overall, by construction, $\mathcal{B}_\Delta$ sets $A[i] = \perp$ for every input bit that is not $\mathsf{admissible}$ or for which $\mathsf{Evt}_{\text{bckt}}(i, \delta)$ does not occur. Then combining the analysis of condition 1 and 2, we conclude that, with probability $1 - 1/n$, $\mathcal{B}_\Delta$ satisfies that for all but a small constant fraction of the input bits, it outputs estimated differences with small bounded errors, and for the rest, it outputs $\perp$. This concludes the lemma. □

4.7.2 Construction and Analysis of the Combined Algorithm $\mathcal{B}$ (Proof of Theorem 4.4)

We now present the algorithm $\mathcal{B}$ that combines algorithms $\mathcal{B}_\Delta$ and $\mathcal{C}'$, and prove Proposition 4.4. On input a constant $\alpha^* \in (0, 1]$ and $(G, y, y^1, \cdots, y^T)$, $\mathcal{B}$ works with parameters in the following ranges so that the premises of Lemma 4.5 and Lemma 4.1 with $r = 1$ are satisfied. For a sufficiently large N,

$$
\begin{aligned}
&\eta = 1 - (1-\varepsilon)^{d-1}, && \mu = (\rho - (\rho+2)\eta)/16, \qquad \alpha = \alpha^*/(2d) \\
&\gamma = \varepsilon^d/5d && \implies \quad \gamma \le \min((1-2\varepsilon)/4, \varepsilon/4) \\
&\varepsilon \le \rho/(d(\rho+2)+1/2) && \implies \quad \eta < \rho/(\rho+2) \le 1/3 \text{ and } \mu \ge \varepsilon/32 \\
&D > \frac{Nd}{\varepsilon^2\alpha^*}\log\left(\frac{2}{\varepsilon}\right), && \implies \quad D > 1/(\alpha^2 d) \text{ and } D > 4\log(4/\gamma)/\mu^2 + 12 \\
&T > Nd^2\log(Dn)/\varepsilon^{2d} && \implies \quad T > 5\log n/\gamma^2 \text{ and } T > 8(d + \log(Dn^2))/\varepsilon^d
\end{aligned}
$$

$\mathcal{B}$ proceeds in the following two stages:

Stage 1–Estimate the difference: Estimates the difference $\Delta^t = x \oplus x^t$ for every instance t. Invoke algorithm $\mathcal{B}_\Delta$ with input $(\mu, G, y, y^1, \cdots, y^T)$, which returns an array A. Let $Q \subseteq [n]$ be the set of input bits i s.t. $A[i] \neq \perp$.

By Lemma 4.5, with high probability, it holds that for every $i \in Q$, $A[i] = \{\tilde{Delta}_i^t\}$ contains the estimated differences for i with γ-bounded errors and $|Q| > (1-\alpha)n$.

Stage 2–Recover an approximate solution: Apply algorithm $\mathcal{C}'$ on the sub-problem induced by Q with the estimated differences returned in Stage 1. More precisely, let $\hat{G}$ be the graph induced by Q: $\hat{G}$ includes all the output vertexes that have all d neighbors in Q, and all the input vertexes in Q that are connected to one of these output vertexes. Let $\hat{y}$ be the projection of y on the output vertexes of $\hat{G}$, and $\hat{y}^t$ be the projection of y^t for every $t \in [T]$; similarly let $\hat{\Delta}^t$ the projection of $\tilde{Delta}^t$ on the input vertexes of $\hat{G}$. Invoke $\mathcal{C}'$ with input $(P, \hat{G}, \hat{y}, \{\hat{y}^t\}, \{\hat{\Delta}^t\})$, which returns $\hat{x}$. Finally, set string $\tilde{x} \in \{0,1\}^n$ as follows: for every input vertex i in $\hat{G}$, set $\tilde{x}_i$ to the value of the corresponding bit in $\hat{x}$; for the rest of the bits that do not appear in $\hat{G}$, set $\tilde{x}_i$ to a randomly chosen bit. Output $\tilde{x}$.

We show that with probability at least $1 - 3/n$, $\tilde{x}$ satisfies a $(1-\alpha^*)$ fraction of the output constraints, which concludes Proposition 4.4.

Claim 4.9. *Let all parameters be set as above. For every $n \in \mathbb{N}$, the probability that $\mathcal{B}(\alpha^*, G, P, y, \{y^t\})$ outputs $\tilde{x}$ s.t. $f(\tilde{x})$ matches y at a $(1-\alpha^*)$ fraction of the output bits is at least $(1 - 1/3n)$.*

Proof. Since the graph $\hat{G}$ is induced by Q, by Lemma 4.5, with probability $1-1/n$, $\{\hat{\Delta}^t\}$ has γ-bounded errors w.r.t. the true differences $\{\Delta^t\}$ and $|Q| \geq (1-\alpha)n$. Then by Lemma 4.1, $\mathcal{C}'$ returns a valid pre-image of $\hat{y}$ with probability $1-1/n$.

It only remains to show that $\hat{y}$ contains at least a $(1-\alpha^*)$ fraction of the output bits. Since the set $\bar{Q} = [n] \setminus Q$ contains at most an α fraction of the input bit, by Claim 4.5, when $D > 1/(\alpha^2 d)$, the fraction of output bits that are connected with any input bit in $\bar{Q}$ is bounded by $2d\alpha$ except with probability $\exp(-(2\alpha^2 dD - 1)n)$. Thus, the fraction of output bits that are only connected with input bits in Q is at least $1-2d\alpha$. Thus, the graph $\hat{G}$ contains at least a $(1-2d\alpha) = 1-\alpha^*$ fraction of the output vertexes of G, as $\alpha = \alpha^*/2d$.

Overall, by a union bound, $\hat{x}$ satisfies at least a $1-\alpha^*$ fraction of the output constraints, with probability at least $1-3/n$. □

5 NP-hard CSPs are easy when given correlated instances

In this section, we show that CSP instances that are generated in the LCLC (label cover and long code) paradigm are easy under correlation, despite being the hardest known instances in terms of approximation. In particular, we show that the satisfiable 3SAT instances generated by Håstad's reduction, while being "hard to invert" even approximately, become "easy to invert" if given one additional correlated instance.

Håstad showed a reduction that maps any NP language to 3SAT, such that yes instances are mapped to satisfiable 3SAT instances, and such that given a satisfiable instance it is NP-hard to find an assignment that satisfies even a $\frac{7}{8}+\varepsilon$ fraction of the clauses. We show that given a satisfiable instance φ with, in addition, a correlated instance φ', finding a satisfying assignment for φ becomes easy.

Theorem 5.1. *Let $L \in NP$ and let $\delta, \varepsilon > 0$. There is a polynomial time reduction H (due to Håstad) mapping instances of L to instances of* 3SAT *such that given a yes instance $\varphi \in$ 3SAT it is NP-hard to find an assignment x satisfying more than $\frac{7}{8}+\delta$ fraction of the clauses of φ. Nevertheless, given also an auxiliary correlated instance $\varphi' \in$ 3SAT there is a polynomial time algorithm that finds a satisfying assignment for φ with high probability over the choice of φ'. The correlated instance φ' is generated as follows*

1. *Let $x \in \{0,1\}^n$ be a satisfying assignment for φ. Flip each bit of x independently with probability ε, obtaining x'.*
2. *Let φ' be the* 3SAT *formula obtained by deleting from φ all of the clauses that are unsatisfied by the assignment x'.*

Proof. The reduction H is exactly Håstad's reduction [Hås01], which we describe next. First observe that Håstad's theorem implies our first assertion of NP-hardness. The reduction first maps the instance of L to a LabelCover instance, namely, a bipartite graph $G = (U, V, E)$ such that each edge $uv \in E$ is decorated by a constraint $\pi_{uv} : \Sigma_1 \to \Sigma_1$ and Σ_1, Σ_2 are some finite alphabets. This part of the reduction has the property that if the initial instance is satisfiable then there is a labeling of the vertexes that satisfies all of the constraints; and otherwise every labeling satisfies all but a δ fraction of the constraints. Next, replace each vertex $u \in U$ by a cloud of 2^{m_1} Boolean variables for $m_1 = |\Sigma_1|$ and each vertex $v \in V$ is replaced by a cloud of 2^{m_2} variables where $m_2 = |\Sigma_2|$. For each edge $uv \in E$ we place a collection of 3SAT clauses on the variables as follows. Let $f : \{0,1\}^{m_1} \to \{0,1\}$ and $g : \{0,1\}^{m_2} \to \{0,1\}$ be Boolean functions whose values are

the assignment to the cloud of u and v respectively. The clauses are described by the following randomized process

1. Choose δ randomly in some set of constant number of small values for δ.
2. Choose a random $x \in \{0,1\}^{m_1}$ and independently choose $y_1 \in \{0,1\}^{m_2}$
3. For each coordinate $j \in [m_2]$ do: if $x[\pi_{uv}(j)] = 0$ choose $y_2[j] = 1 - y_1[j]$ and if $x[\pi_{uv}(j)] = 1$ choose $y_2[j] = y_2[j]$ with probability $1 - \delta$ and $y_2[j] = 1 - y_2[j]$ with probability δ.
4. Accept if $f(x) \vee g(y_1) \vee g(y_2)$.

Satisfying assignments for φ all have the following "blockwise-dictatorship-form". Each cloud is assigned a dictatorship Boolean function, i.e. a function of the form $h(x_1, \ldots, x_m) = x_i$ for some i such that this collection of i's form a perfectly satisfying labeling for the LabelCover instance G. Let A be a satisfying assignment for φ. It is best to describe A as a collection $\{f_u : \{0,1\}^{m_1} \to \{0,1\}\}_u \cup \{g_v : \{0,1\}^{m_2} \to \{0,1\}\}_v$ of Boolean functions, one per cloud. Let A' be a perturbation of A used for generating φ'. We will use the auxiliary input φ' to recover an assignment very close to A, that satisfies almost all of the clauses in φ (and in particular, much more than 7/8 fraction).

The crucial extra information we get from φ' is an approximation about which clauses are satisfied in φ by *only one literal*. These clauses are much more likely to become unsatisfiable (because they require one bit flip as opposed to two or more bit flips). This information will help us uncover a labeling for G and therefore an assignment satisfying most of φ.

For a fixed edge uv, let $f : \{0,1\}^{m_1} \to \{0,1\}$ and $g : \{0,1\}^{m_2} \to \{0,1\}$ be Boolean functions representing the assignment to the clouds of u and v respectively. A clause on these two clouds has the form $f(x)^{b_1} \vee g(y_1)^{b_2} \vee g(y_2)^{b_3}$ (notation: we let $f(x)^b = f(x)$ if $b = 0$ and $f(x)^b = \neg f(x)$ if $b = 1$)[9]. In a dictatorship-form assignment, this clause is satisfied by *only one literal* in two cases. Either when $f(x)^{b_1} = 0$, or when $f(x)^{b_1} = 1$ with probability $(1 - \delta)/2$ (over the choices of the randomized process generating y_1, y_2 conditioned on x). This means that if $f(x_1, \ldots, x_{m_1})^{b_1} = x_i^{b_1}$ then this occurs either when $x_i^{b_1} = 0$, or when $x_i^{b_1} = 1$ with probability slightly less than half. So if we know that the clause is satisfied by one literal only it is more likely that $x_i^{b_1} = 0$, and this suggests the following algorithm.

For every $i \in [m_1]$ let $C_0(i)$ be all clauses that contain $f(x)$ for an x with $x_i = 0$ and let $C_1(i)$ be all clauses containing $\neg f(x)$ for an x with $x_i = 1$. Let $C_{violated}$ be the collection of clauses that belong to φ but not φ', so they are violated by the perturbed assignment. Let $p(i)$ be the fraction of clauses in $C_{violated}$ that are in $C_0(i) \cup C_1(i)$.

We expect $p(i) \approx 3/4$ (ignoring ε, δ fractions) for the correct label i, and $p(i) \approx 1/2$ for all other labels. It is not hard to see that this random variable is quite concentrated which means that by taking i to be the label with the highest $p(i)$ we will succeed with probability close to 1.

□

6 Inverting Randomized Encoding

A randomized encoding [IK02] allows to represent a "complex" (deterministic) function f by a "simple" (randomized) function $\hat{f}$. The output $\hat{f}(x; r)$ of the new function encodes the output $f(x)$ of the original function, and no more information—the output distribution $\hat{f}(x, U)$ can be

[9] The negations arise from folding, a subtlety that was glossed over in the presentation above.

emulated using only $f(x)$. One measure of complexity that has been studied in the literature is the depth of the circuit computing the function. In [AIK04], Applebaum, Ishai, and Kushilevitz presented a construction of randomized encoding for representing any $\mathcal{NC}^1$ computable function by a $\mathcal{NC}^0$ function, with only output locality 4. [10] Since many candidate one-way functions, based on most number theoretic or algebraic intractability assumptions commonly used in cryptography (such as factoring and discrete logarithm), are computable in $\mathcal{NC}^1$, the AIK randomized encoding allows to represent these one-way functions using very "simple", $\mathcal{NC}^0_4$ (randomized) functions, while retaining the one-wayness (as the output distribution of the new function reveals no more information other than the output of the original one-way function). However, in this section, we show that functions produced by the AIK randomized encoding are easy to invert given two correlated instances, $\hat{f}(x;r), \hat{f}(x';r)$, that encodes the outputs of two different strings using the same random coins. This stands in contrast with the previous observation that the AIK represented one-way functions are still one-way.

Below we first review the definition of the randomized encoding and the AIK construction, and then show how to invert this particular randomized encoding, using correlated instances.

6.1 Definition of Randomized Encoding and Branching Programs

Randomized Encoding: Below we recall the definition of a *prefect randomized encoding* for efficiently computable functions from [AIK04].

Definition 6.1 ([AIK04])**.** *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomial time computable function, and $l(n)$ an output length function such that $|f(x)| = l(|x|)$ for every $x \in \{0,1\}^*$. We say that $\hat{f} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ is a perfect randomized encoding of f, if it satisfies the following:*

Length Regularity: *There exists polynomially-bounded and efficiently computable functions $m(n)$, $s(n)$, such that for every $x \in \{0,1\}^n$ and $r \in bitset^{m(n)}$, we have $|\hat{f}(x,r)| = s(n)$. Call m and s the randomness-length and output-length polynomials.*

Efficient Evaluation: *There exists a polynomial-time evaluation algorithm that on input $x \in \{0,1\}^*$ and $r \in \{0,1\}^{m(|x|)}$ outputs $\hat{f}(x,r)$.*

Perfect Correctness: *There exists a polynomial-time algorithm C, called a decoder, such that, for every input $x \in \{0,1\}^n$, $\Pr[C(\hat{f}(x, U_{m(n)})) = f(x)] = 1$.*

Perfect Privacy: *There exists a randomized polynomial-time algorithm S, called a simulator, such that, for every input $x \in \{0,1\}^n$, $\|S(f(x)) - \hat{f}(x, U_{m(n)})\| = 0$.*

When saying that a uniform encoding $\hat{f}$ is in a (uniform) circuit complexity class, it means that its evaluation algorithm can be implemented by circuits in this class. For instance, we say that $\hat{f}$ is in $\mathcal{NC}^0_d$ if there exists a polynomial-time circuit generator G such that $G(1^n)$ outputs a d-local circuit computing $\hat{f}(x;r)$ on all $x \in \{0,1\}^n$ and $r \in \{0,1\}^{m(n)}$.

Below by randomized encoding, we mean perfect randomized encodings.

Branching Programs: A basic Branching Program (BP) for n input variables $x_1, \cdots x_n$ is defined by a tuple $BP = (G, \phi, src, t_A, t_R)$. $G = (V, E)$ is a directed acyclic graph with three distinguished nodes, a source node src and two sink nodes t_A for Accept and t_R for Reject. ϕ is a

[10] In fact, the AIK randomized encoding can handle functions computable by any polynomial-sized mod-2 BPs. In this work, we focus on $\mathcal{NC}^1$ only.

labelling function assigning every edge a label, which is 1, a positive literal x_i or a negative literal $\bar{x}_i$. We consider *layered BPs* of length ℓ. Here the nodes are partitioned into $\ell+1$ sets $V_1, \cdots V_{\ell+1}$ and edges only go from one layer to the next. The start node is in layer 1 and the sink nodes in layer $\ell+1$. Every layer $\theta \in [\ell]$ (except from the last one) depends on an input variable x_{i_l}, meaning that all edges leaving layer θ are labelled as either x_{i_l}, $\bar{x}_{i_l}$ or 1.

Additionally, we assume the following canonical form of the layered BPs:

Property 1: Except from the last layer, every node has two out-going edges, one is labeled with a positive literal x_i, and the other with the corresponding negative literal $\bar{x}_i$. (In particular, no edge is labelled by 1.)

Property 2: Except from the first and last layers, every node has two in-coming edges, one is labeled with a positive literal x_i, and the other with the corresponding negative literal $\bar{x}_i$.

In such a canonical BP, every assignment $w \in \{0,1\}^n$ to the input variables $\{x_1, \cdots, x_n\}$ induces a *unique* path in G from the source node to one of the node in the last layer, and it is accepted if and only if the path it induces leads to t_A.

Remark 6.1. *We note that the work of [AIK04] considered a more general definition of BPs, where an assignment w may induce a subgraph G_w (by including all edges $e \in E$ such that $\phi(e)$ is satisfied by w) instead of just a unique path. Furthermore, in [AIK04] BPs may be assigned different semantics: in a non-deterministic BP, an assignment w is accepted if there is at least one path from src to t_A; in a (counting) mod-2 BP, the BP computes the number of paths from src to t_A modulo 2. A construction of randomized encoding for mod-2 BPs is presented in [AIK04], which can also be extended to handle non-deterministic BPs.*

When focusing on canonical BPs as described above, since every x induces a unique path, both non-deterministic BP and mod-2 BP accept the same set of inputs. Therefore, we do not distinguish between the two semantics, and the construction of [AIK04] directly applies to BPs with the canonical form.

In the rest of the paper, by BP, we mean BPs with the canonical form. We say that the size of a BP is the number of nodes in G, the length is the length of the longest path in G, and the width is the maximum number of vertexes in any layer. We show that it follows essentially from Barrington's theorem that every function in $\mathcal{NC}^1$ can be implemented using a canonical BP of polynomial size.

Theorem 6.1 (Implied by Barrington's Theorem)**.** *Every Boolean function $f : \{0,1\}^* \to \{0,1\}$ in $\mathcal{NC}^1$ can be computed by a family of polynomial-sized canonical BPs.*

Proof. The Barlington's Theorem showed that Boolean function that can be computed by a circuit with depth d can also be computed using a layered BP of length $\ell = 4^d$ and width 5. Furthermore, for every layer $\theta \in [\ell]$ depending on an input variable x_{i_θ}, all edges induced by an assignment $x_{i_\theta} = 1$ form a permutation from nodes in layer θ to nodes in layer $\theta+1$, and all edges induced by an assignment $x_{i_\theta} = 0$ form another permutation. Such layered BPs almost have the canonical form except that some nodes may have only one out-going edge (or only one in-coming edge) labelled by 1.

This problem can be easily fixed by simply duplicating the nodes in every layer, except from the first and the last layers; let V^0_θ be the original set of nodes in layer θ, and V^1_θ the duplicated set. Every out-going edge from V^0_θ with label $\bar{x}_{i_\theta}$ or 1, which is originally connected to a node in $V^0_{\theta+1}$, is now re-wired to the corresponding node in $V^1_{\theta+1}$ and labelled by $\bar{x}_{i_\theta}$; similarly, every such out-going

edge from V^1_θ is re-wired to the corresponding node in $V^0_{\theta+1}$ and labelled by $\bar{x}_{i_\theta}$. (Edges from V^0_θ labelled by x_{i_θ} or 1 remain to connect to nodes in $V^0_{\theta+1}$, and such edges from V^1_θ are connected to corresponding nodes in V^1_θ.) After this modification, every edge with label 1 is duplicated to two edges with label x_{i_θ} and $\bar{x}_{i_\theta}$ respectively. Thus, the resulting BP has the canonical form. □

6.2 The AIK Randomized Encoding

The construction of AIK randomized encoding in $\mathcal{NC}^0_4$ for functions computable in $\mathcal{NC}^1$ relies on the construction of degree-3 randomizing polynomials for mod-2 BPs from [IK02], and a locality reduction technique. Below we start with reviewing each part.

Degree-3 Randomizing Polynomials: Let $BP = (G, \phi, s, t_A, t_R)$ be a BP of length ℓ and size η computing Boolean function $f : \{0,1\}^n \to \{0,1\}$, that is, $f(x) = 1$ if and only if the unique path induced by x in G leads to t_A, or equivalently, the number of paths from src to t_A induced by x equals to 1 mod 2. Fix some topological ordering over nodes in G, where the source node is labelled by 1 and t_A is labelled by η. Let $A(x)$ be the $\eta \times \eta$ adjacency matrix of the graph G, where $A(x)_{ij}$ is non-zero if and only if there is an edge from node i to j in G and equals to the labelling $\phi((i,j))$ on that edge. Hence, $A(x)$ contains the constant 0 on and below the main diagonal, and degree-1 polynomials in a single input variable x_i above the main diagonal. Define $L(x)$ as the submatrix of $A(x) - I$ obtained by deleting the first column and the last row. As before, each entry of $L(x)$ is a degree-1 polynomial in a single input variable x_i; moreover, $L(x)$ contains the constant -1 in each entry of its second diagonal (the one below the main diagonal) and the constant 0 below this diagonal. The following fact was shown in [IK02].

Fact 6.1. *$f(x) = \det(L(x))$, where the determinant is computed over $GF(2)$.*

Therefore, to hide the input x while still allowing the computation of $f(x)$, it suffices to randomize the matrix $L(x)$ to a random matrix with the same determinant. Towards this, let $r^{(1)} = \{{r^{(1)}}_{ij}\}_{1 \le i < j \le \eta-1}$ and $r^{(2)} = \{{r^{(2)}}_k\}_{1 \le k \le \eta-2}$ be bit vectors of size $\Gamma = (\eta-1)(\eta-2)/2$ and $(\eta-2)$ respectively. Let $R_1(r^{(1)})$ be an $(\eta-1) \times (\eta-1)$ matrix with 1's on the main diagonal, 0's below it, and ${r^{(1)}}_{ij}$ on entry (i,j) above the diagonal. Let $R_2(r^{(2)})$ be an $(\eta-1) \times (\eta-1)$ matrix with 1's on the main diagonal, ${r^{(2)}}_k$ on entries $(k, \eta-1)$ in the last column, and 0's in each of the remaining entries.

Fact 6.2. *Let M, M' be $(\eta-1) \times (\eta-1)$ matrices that contain the constant -1 in each entry of their second diagonal and the constant 0 below this diagonal. Then $\det(M) = \det(M')$ if and only if there exists $r^{(1)}$ and $r^{(2)}$ such that $R_1(r^{(1)}) M R_2(r^{(2)}) = M'$.*

Therefore, to randomize $L(x)$ it suffices to sample $r^{(1)}$ and $r^{(2)}$ at random and output $R_1(r^{(1)})L(x)R_2(r^{(2)})$. As shown in [IK02, AIK04], this indeed yields a randomized encoding.

Lemma 6.1 ([IK02, AIK04])**.** *Let BP be a branching program computing the Boolean function f. Define a degree-3 function $RP(x; r^{(1)}, r^{(2)})$ whose output contains the $\Gamma = (\eta-1)(\eta-2)/2$ entries above the diagonal of the matrix $M = R_1(r^{(1)})L(x)R_2(r^{(2)})$. RP is a randomized encoding for BP.*

Every entry in M is a degree-3 polynomial in variables $\{x_i\}$, $\{{r^{(1)}}_{ij}\}$ and $\{{r^{(2)}}_k\}$.

$$M_{ij}(x; r^{(1)}, r^{(2)}) = \Sigma_{t,k} \left(R_1(r^{(1)})_{it} L(x)_{tk} R_2(r^{(2)})_{kj} \right) = \Sigma_{t,k} \left(R_1(r^{(1)})_{it} (A(x)_{t,k+1} - I_{t,k+1}) R_2(r^{(2)})_{kj} \right)$$

Denote by $\{T_{itkj}\}$ the set of degree-3 monomials in M_{ij}'s.

$$T_{itkj} = T_{itkj}(x, r^{(1)}, r^{(2)}) = R_1(r^{(1)})_{it} L(x)_{tk} R_2(r^{(2)})_{kj}$$

Although the degree of each bit of output of RP is low, the locality is not, in particular every output bit depends on many monomials as above and many variables.

Reducing Locality: To reduce a degree-3 encoding into one in $\mathcal{NC}^0_4$. [AIK04] constructed for any degree-d function (where d is a constant greater than 1) a locality $d+1$ randomized encoding. The idea is to represent a degree-d polynomial as a sum of monomials, each having locality d, and randomize this sum as follows: Let $g : \{0,1\}^n \to \{0,1\}$ be a Boolean function with degree d; then $g(x) = T_1(x) + \cdots, T_k(x)$ over $GF(2)$ where $\{T_j\}$ are Boolean functions representing the monomials in f and each have locality d. The local encoding $LR : \{0,1\}^n \times \{0,1\}^{2k-1} \to \{0,1\}^{2k}$ is define by:

$$\begin{aligned} LR(x; (r_1, \cdots, r_k, r'_1, \cdots, r'_{k-1})) \quad = \quad & (T_1(x) + r_1, \cdots, T_k(x) + r_k, \\ & -r_1 - r'_1, r'_1 - r_2 - r'_2, \cdots, r'_{k-2} - r_{k-1} - r'_{k-1}, r'_{k-1} - r_k) \end{aligned}$$

As shown in [AIK04], LR is a randomized encoding for f and has locality $d+1$ when $d > 1$.

Locality-4 Randomized Encoding Combining degree-3 randomizing polynomials with the locality reduction technique gives a locality-4 randomized encoding for functions $f : \{0,1\}^n \to \{0,1\} \in \mathcal{NC}^1$. By Barrington's Theorem, every Boolean function in $\mathcal{NC}^1$ can be computed by a (uniform) family of BPs $\{BP_n\}_{n\in\mathbb{N}}$ of polynomial size $\eta(n)$. Let $RP_n : \{0,1\}^n \times \{0,1\}^\Gamma \times \{0,1\}^{\eta-2} \to \{0,1\}^\Gamma$ be the randomizing polynomial for BP_n. Since each output bit M_{ij} of RP_n has degree-3, applying the locality reduction on it yields a locality-4 encoding LR_{ij}, where every monomial T_{itkj} in M_{ij} is hidden using random strings $({r^{(3)}}_{ij}, {r^{(4)}}_{ij})$. In particular,

$$LR_{ij}(x, r^{(1)}, r^{(2)}; ({r^{(3)}}_{ij}, {r^{(4)}}_{ij})) = \left\{ ({T_{itkj}} + {r^{(3)}}_{itkj}) \right\}_{tk}, \ \left\{ {r^{(4)}}_{ij,t+k-1}, {r^{(3)}}_{itkj} - {r^{(4)}}_{ij,t+k} \right\}$$

where ${r^{(3)}}_{itkj}$ denote the $t+k^{\text{th}}$ bit in ${r^{(3)}}_{ij}$. The final randomized encoding $\hat{f}(x; r^{(1)}, r^{(2)}, (r^{(3)}, r^{(4)}))$ outputs the concatenation of all the local encodings $\{LR_{ij}(x, r^{(1)}, r^{(2)}; ({r^{(3)}}_{ij}, {r^{(4)}}_{ij}))\}_{ij}$.

It was shown in [AIK04] that the composition and concatenation of randomized encodings is still a randomized encoding. Thus $\hat{f}$ is a randomized encoding of f and is computable in $\mathcal{NC}^0_4$. If the original function f have multiple output bits, one can obtain a randomized encoding by concatenating the randomized encodings for functions $f_1, \cdots, f_l$, where f_i outputs the i^{th} output bit of f.

In general, the randomized encoding can be instantiated with any construction of BPs for $\mathcal{NC}^1$ functions. In this work, we will focus on the special case of using canonical BPs.

6.3 Inverting AIK Randomized Encoding with Shared Randomness

In this section, we show that for every function $f \in \mathcal{NC}^1$, the AIK encoding $\hat{f}$ is easy to invert given two instances $\hat{f}(w; r)$ and $\hat{f}(w'; r)$ using shared randomness. More specifically, for arbitrary $w \neq w'$, partial information of the assignments can be recovered, and when w' is derived from w probabilistically by flipping each bit of w with a small constant probability ε, w and w' can be recovered completely.

Theorem 6.2. *For every function f in $\mathcal{NC}^1$, let $\hat{f}$ be the AIK randomized encoding of f using $m(\cdot)$-bit randomness. Fix any $\varepsilon \in (0, 1/2]$ and any constant $\alpha > 0$. There is a polynomial time algorithm, such that for every function $f \in \mathcal{NC}^1$, $n \in \mathbb{N}$, $w \in \{0,1\}^n$, and $r \in \{0,1\}^{m(n)}$, the algorithm on input $(1^n, f^n, y = \hat{f}(w, r), y' = \hat{f}(w', r))$ with $w' \xleftarrow{\$} \mathcal{D}_\varepsilon(w)$, inverts $f(w)$ and $f(w')$ with probability $1 - n^{-\alpha}$.*

Towards proving the theorem, we first consider the simpler case of Boolean functions, and then discuss about non-Boolean functions.

6.3.1 Handling Boolean Functions

Given a Boolean function $f : \{0,1\}^* \to \{0,1\}$ in $\mathcal{NC}^1$, let $\hat{f} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be the AIK randomized encoding of f that on input a n-bit string x uses $m(n)$ random coins and outputs a $s(n)$-bit encoding, and $\{BP_n\}_{n\in\mathbb{N}}$ the underlying family of canonical BPs for computing f of size $\eta(n) < s(n)$ and length $\ell(n)$. Recall that each layer $\theta \in [\ell(n)]$ in the branching program BP_n depends on an input variable x_{i_θ}. We order the input variables according to the sequence in which they appear in the branching program (from the first layer to the last), $x_{\pi(1)}, \cdots, x_{\pi(n)}$. (Each variable x_i must appear in at least one layer; thus π is a permutation). Given the randomized encoding of any two different assignments w and w' of x using shared randomness r, we show that partial information of w and w' can be recovered.

Lemma 6.2. *For every Boolean function f in $\mathcal{NC}^1$, let $\hat{f}$ be the AIK randomized encoding of f using $m(\cdot)$-bit randomness, and $\{BP_n\}$ the underlying canonical BPs. There is an algorithm $\mathcal{D}$ such that for every $f \in \mathcal{NC}^1$, $n \in \mathbb{N}$, $w, w' \in \{0,1\}^n$, and $r \in \{0,1\}^{m(n)}$, it holds that $\mathcal{D}$ on input $(1^n, f^n, y = \hat{f}(w, r), y' = \hat{f}(w', r))$ recovers $(w_{\pi(u^*-1)}, w_{\pi(u^*-2)}, \cdots, w_{\pi(1)})$ and $(w'_{\pi(u^*-1)}, w'_{\pi(u^*-2)}, \cdots, w'_{\pi(1)})$, where π is a permutation describing the order of sequence in which input variables appear in BP_n and u^* is the largest $u \in [n]$ such that $w_{\pi(u)} \neq w'_{\pi(u)}$.*

Lemma 6.2 considers two arbitrary assignments w and w'. When the assignments are correlated in the sense that w' is derived by flipping each bit in w with probability ε, the probability that $n - u^* = a$ for any $a \in [n]$ is $(1-\varepsilon)^a\varepsilon$. Thus, for every constant α, it holds that except with probability $1/n^\alpha$, the number of variables whose values are not recovered is $n - u^* + 1 \leq \frac{\alpha \log n}{-\log(1-\varepsilon)} = O(\log n)$; in this case, the algorithm $\mathcal{D}$ in proof of Lemma 6.2 recovers all but $O(\log n)$ bits of w and w'.

Corollary 6.1. *For every Boolean function f in $\mathcal{NC}^1$, let $\hat{f}$ be the AIK randomized encoding of f using $m(\cdot)$-bit randomness, and $\{BP_n\}$ the underlying canonical BPs. Fix any $\varepsilon \in (0, 1/2]$ and any constant $\alpha > 0$. There is a polynomial time algorithm $\mathcal{D}$ such that for every $f \in \mathcal{NC}^1$, $n \in \mathbb{N}$, $w \in \{0,1\}^n$, and $r \in \{0,1\}^{m(n)}$, $\mathcal{D}$ on input $(1^n, f^n, y = \hat{f}(w, r), y' = \hat{f}(w', r))$ with $w' \xleftarrow{\$} \mathcal{D}_\varepsilon(w)$, outputs all but $\frac{\alpha \log n}{-\log(1-\varepsilon)}$ bits of w and w', with probability $1 - n^{-\alpha}$.*

Proof of Lemma 6.2. Fix a Boolean function $f \in \mathcal{NC}^1$ and one $n \in \mathbb{N}$. Let $BP = (G, \phi, src, t_A, t_R)$ be the canonical BP for computing f on n bit inputs; BP has size $\eta < s(n)$ and can be constructed efficiently from the description of f in $\text{poly}(\eta) < \text{poly}(s(n))$ steps. Let $A(x)$ the $\eta \times \eta$ adjacency matrix of G, and $L(x)$ the $(\eta-1)\times(\eta-1)$ matrix that equals to $A(x) - I$ removing the first column and last row. Recall that the randomized polynomial RP of BP consists of the $\Gamma = (\eta-1)(\eta-2)/2$ entries above the diagonal of matrix M computed as,

$$M_{ij}(x; r^{(1)}, r^{(2)}) = \Sigma_{t,k} \left(R_1(r^{(1)})_{it} L(x)_{tk} R_2(r^{(2)})_{kj} \right) = \Sigma_{t,k} T_{itkj}(x; r^{(1)}, r^{(2)}),$$

and the randomized encoding $\hat{f}$ consists of the local encodings LR_{ij} of each M_{ij}; in particular,

$$LR_{ij}(x, r^{(1)}, r^{(2)}; ({r^{(3)}}_{ij}, {r^{(4)}}_{ij})) \text{ includes a term } (T_{itkj} + {r^{(3)}}_{itkj}) \text{ for every } t, k$$

Before proceeding to describe the algorithm for inverting AIK randomized encoding, we first describe three useful subroutines.

Subroutine $\mathcal{D}_1$ for determining differences The first subroutine $\mathcal{D}_1$ described in Figure 6 determines whether the i^{th} bits of two assignments w and w' differ or not, given a pair of AIK encoding using shared randomness $y = \hat{f}(w, r), y' = \hat{f}(w', r)$. We in-line some analysis in the algorithm in square brackets.

Algorithm $\mathcal{D}_1$ for estimating the difference between w and w'

On input 1^n, BP, $y = \hat{f}(w, r)$, $y' = \hat{f}(w', r')$ and $i \in [n]$ proceed as follows:

1. Find any edge (t, k) in G with label x_i, that is $A_{tk} = x_i$. [a]
2. Consider the monomial

$$T_{tt,k-1,k-1}(x) = R_1(r^{(1)})_{tt} L(x)_{t,k-1} R_2(r^{(2)})_{k-1,k-1} = 1 \times A_{tk} \times 1 = x_i$$

   Find the values corresponding to term $T_{tt,k-1,k-1} + {r^{(3)}}_{tt,k-1,k-1}$ in y and y', which are,

$$v = T_{tt,k-1,k-1}(w) + {r^{(3)}}_{tt,k-1,k-1} = w_i + {r^{(3)}}_{tt,k-1,k-1}$$
$$v' = T_{tt,k-1,k-1}(w') + {r^{(3)}}_{tt,k-1,k-1} = w'_i + {r^{(3)}}'_{tt,k-1,k-1}$$

3. Determine whether $w_i = w'_i$: Set $\tilde{D}elta_i = 0$ if $v = v'$ and 1 otherwise.

[*Remark: The computation of $\tilde{D}elta_i$ depends only on random coins ${r^{(3)}}_{tt,k-1,k-1}$ and ${r^{(3)}}'_{tt,k-1,k-1}$. When they are equal (which happens when y and y' are generated using identical random coins), $w_i = w'_i$ if and only if $v_1 = v_2$, and thus the difference between w_i and w'_i is recovered correctly.*]

[a] By Property 1 of the canonical BPs and the fact that x_i must be used in some layer, such an edge must exist.

Figure 6: When given a pair $y = \hat{f}(w, r), y' = \hat{f}(w', r)$ and i, whether $w_i \neq w'_i$ can be determined.

It follows directly from the construction that

Claim 6.1. *For every $w, w' \in \{0,1\}^n$, and $r \in \{0,1\}^{m(n)}$, $\mathcal{D}_1(1^n, BP, y = \hat{f}(w, r), y' = \hat{f}(w', r), i)$ outputs $\Delta_i = w_i \oplus w'_i$.*

Subroutine $\mathcal{D}_2$ for recovering $r^{(1)}$ values If layer $\theta \in [\ell]$ depends on variable x_{i_θ} and its value flips from from w to w', the second subroutine recovers all $r^{(1)}$ values associated with nodes in layer θ. More specifically, for every node t in layer θ, the $r^{(1)}$ values associated with t are $\{{r^{(1)}}_{i,t}\}_{i<t}$; denote them as $r^{(1)}(t)$. Similarly, the $r^{(1)}$-values associated with layer θ include all $r^{(1)}$-values associated with every node in this layer $\{{r^{(1)}}_{i,t}\}_{i<t, t \in V_\theta}$; denoted as $r^{(1)}(\theta)$ (recall that V_θ is the set of nodes in layer θ). When the value of x_{i_θ} flips from w to w', subroutine $\mathcal{D}_2$ described in Figure 7 recovers all $r^{(1)}$-values associated with layer θ—in this case, we say layer l is ready. Formally,

It follows from the construction of $\mathcal{D}_2$ that,

Algorithm $\mathcal{D}_2$ for recovering $r^{(1)}$-values

On input 1^n, BP, $y = \hat{f}(w, r)$, $y' = \hat{f}(w', r')$, and $\theta \in [\ell]$, proceed as follows:

For every node t in layer θ, recover the $r^{(1)}$-values associated with t as follows:

1. Find any edge (t, k) in G with label x_{i_θ}, that is $A_{tk} = x_{i_\theta}$. [a]
2. For every $r^{(1)}$ value ${r^{(1)}}_{it}$, consider the monomial

$$T_{it,k-1,k-1}(x) = R_1({r^{(1)}})_{it} L(x)_{t,k-1} R_2({r^{(2)}})_{k-1,k-1} = {r^{(1)}}_{it} \times A_{tk} \times 1 = {r^{(1)}}_{it} x_{i_\theta}$$

Find the values corresponding to term $T_{it,k-1,k-1} + {r^{(3)}}_{ij,k-1,k-1}$ in y and y', which are,

$$v = T_{it,k-1,k-1}(w) + {r^{(3)}}_{it,k-1,k-1} = {r^{(1)}}_{it} w_{i_\theta} + {r^{(3)}}_{it,k-1,k-1}$$
$$v' = T_{it,k-1,k-1}(w') + {r^{(3)}}'_{it,k-1,k-1} = {r^{(1)}}'_{it} w'_{i_\theta} + {r^{(3)}}'_{it,k-1,k-1}$$

Set $\tilde{r}^{(1)}_{it}$ to $v - v'$.

Finally, output $\tilde{r}^{(1)}(\theta) = \{\tilde{r}^{(1)}_{it}\}_{i<t, t \in V_\theta}$.
[*Remark: For every i, t, the computation of $\tilde{r}^{(1)}_{it}$ depends only on random coins $({r^{(1)}}_{it}, {r^{(1)}}'_{it})$ and $({r^{(3)}}_{it,k-1,k-1}, {r^{(3)}}'_{it,k-1,k-1})$. If both pairs are equal, when $w_{i_\theta} \neq w'_{i_\theta}$, subtracting v and v' recovers ${r^{(1)}}_{it}$.*]

[a] By Property 1 of canonical BPs, such an edge must exist.

Figure 7: If the value of x_{i_θ} flips in w and w', then the $r^{(1)}$-values $r^{(1)}(\theta)$ of layer θ can be recovered from $y = \hat{f}(w, r), y' = \hat{f}(w', r)$. Then layer θ becomes ready.

Claim 6.2. *For every $n \in \mathbb{N}$, canonical branching program BP on n input variables, $w, w' \in \{0,1\}^n$, and $r \in \{0,1\}^{m(n)}$, if layer θ depends on variable x_{i_θ} and $w_{i_\theta} \neq w'_{i_\theta}$, then $\mathcal{D}_2(1^n, BP, y = \hat{f}(w, r), y' = \hat{f}(w', r), \theta)$ outputs $r^{(1)}(\theta)$.*

Subroutine $\mathcal{D}_3$ for recovering assignments When a layer θ is ready, we show how to recover the assignment $w_{i_{\theta-1}}$ to the input variable $x_{i_{\theta-1}}$ associated with the *previous* layer $\theta - 1$, from y and $r^{(1)}$-values of layer θ. (The assignment $w'_{i_{\theta-1}}$ can be recovered from $w'_{i_{\theta-1}} \oplus \Delta_{i_{\theta-1}}$.) When the value $w_{i_{\theta-1}}$ is recovered, we say that layer $\theta - 1$ is solved. The subroutine $\mathcal{D}_3$ is described in Figure 8.

Claim 6.3. *For every $n \in \mathbb{N}$, canonical branching program BP on n input variables, $w \in \{0,1\}^n$, and $r \in \{0,1\}^{m(n)}$, $\mathcal{D}_3(1^n, BP, y = \hat{f}(w, r), \theta, r^{(1)}(\theta))$ outputs $w_{i_{\theta-1}}$.*

Subroutine $\mathcal{D}_4$ for recovering $r^{(1)}$-values after a layer is solved When a layer $\theta - 1$ is solved, and the next layer θ is ready, we show how to use the assignment $w_{i_\theta - 1}$ and the $r^{(1)}$-values $r^{(1)}(\theta)$ to recover the $r^{(1)}$-values $r^{(1)}(\theta - 1)$ associated with layer $\theta - 1$; then layer $\theta - 1$ becomes ready. The sub-routine is described in Figure 9.

It follows from the construction of algorithm $\mathcal{D}_4$ that

Claim 6.4. *For every $n \in \mathbb{N}$, canonical branching program BP on n input variables, $w \in \{0,1\}^n$, and $r \in \{0,1\}^{m(n)}$, $\mathcal{D}_4(1^n, BP, y = \hat{f}(w, r), \theta, w_{i_\theta - 1}, r^{(1)}(\theta))$ outputs $r^{(1)}(\theta - 1)$.*

Algorithm $\mathcal{D}_3$ for recovering assignments

On input 1^n, BP, $y = \hat{f}(w, r)$, θ and $\tilde{r}^{(1)}(\theta)$ for $0 < \theta \leq \ell$ proceed as follows:

1. Pick any node k in layer θ. Find the "last" in-coming edge (t, k) to k, that is, all other edges (t', k) to k have $t' < t$. It has label $x_{i_{\theta-1}}$ (or $\bar{x}_{i_{\theta-1}}$), that is, $A_{tk} = x_{i_\theta}$ (or $A_{tk} = \bar{x}_{i_{\theta-1}}$) and $A_{t'k} = 0$ for all $t < t'$.

2. Consider following entry in the randomized polynomial.

$$\begin{aligned} M_{t,k-1}(x) &= \Sigma_{t'k'} \left(R_1(r^{(1)})_{tt'} L(x)_{t'k'} R_2(r^{(2)})_{k',k-1} \right) \\ &= \Sigma_{t \leq t', k'=k-1} \left(R_1(r^{(1)})_{tt'} L(x)_{t'k-1} \right) \ ^a \\ &= \Sigma_{t \leq t'} \left(R_1(r^{(1)})_{tt'} \left(A(x)_{t'k} - I_{t'k} \right) \right) \ ^b \\ &= x_{i_{\theta-1}} - {r^{(1)}}_{tk} \quad (\text{or } \bar{x}_{i_{\theta-1}} - {r^{(1)}}_{tk}) \ ^c \end{aligned}$$

   Reconstruct the value of $M_{t,k-1}(w)$ from y and find value $\tilde{r}^{(1)}_{tk}$ in $\tilde{r}^{(1)}(\theta)$. Compute $\tilde{w}_{i_{\theta-1}} = M_{t,k-1}(w) + \tilde{r}^{(1)}_{tk}$ (or $M_{t,k-1}(w) + \tilde{r}^{(1)}_{tk} + 1$).

[*Remark: The computation of $\tilde{w}_{i_\theta - 1}$ depends only on the random coin ${r^{(1)}}_{tk}$ and the output is correct whenever $\tilde{r}^{(1)}_{tk} = {r^{(1)}}_{tk}$.*]

---

[a] This equality follows from the structures of matrixes R_1, and R_2: In the $k-1^{\text{th}}$ column of R_1, only entry $R_2(r^{(2)})_{k-1,k-1} = 1$ and all other entries $R_2(r^{(2)})_{k',k-1} = 0$ for $k' \neq k-1$; R_2 is an upper triangular matrix, and for all $t > t'$ $R_1(r^{(1)})_{t,t'} = 0$.

[b] This equality follows from structure of L: For every entry i, j in L, $L(x)_{ij} = A(x)_{i,j+1} - I_{i,j+1}$.

[c] This equality follows from the fact that $A(x)_{t'k} = 0$ for all $t' > t$ and $A_{tk} = x_{i_\theta}$ (or $A_{tk} = \bar{x}_{i_{\theta-1}}$) and that $R_1(r^{(1)})_{tt} = 1$.

Figure 8: When a layer θ is ready—that is, values $r^{(1)}(V_\theta)$ are known—the assignments to $x_{i_{\theta-1}}$ of the previous layer $\theta - 1$ can be recovered from $y = \hat{f}(w, r)$. Then layer $\theta - 1$ becomes solved.

Putting pieces together With the above four subroutines, the program for inverting AIK randomized encoding is straightforward. Given two randomized encoding $y = \hat{f}(w, r)$ and $y' = \hat{f}(w', r)$ for a $\mathcal{NC}^1$ Boolean function f with a canonical branching program BP, the algorithm $\mathcal{D}$ proceeds in three steps:

Step 1, Recover difference: Compute the difference $\tilde{D}elta$ between w and w' using subroutine $\mathcal{D}_1$; then find the last layer θ^* in BP whose associated input variable $x_{i_{\theta^*}}$ satisfies that $\tilde{D}elta_{i_{\theta^*}} = 1$.

By Claim 6.1, $\mathcal{D}_1$ returns the correct difference $\tilde{D}elta = \Delta = w \oplus w'$. Thus, the input variable $x_{i_{\theta^*}}$ indeed have different assignments $w_{i_{\theta^*}} \neq w'_{i_{\theta^*}}$. Furthermore, order input variables in the sequence they appear in BP, $x_{\pi(1)}, \cdots, x_{\pi(n)}$; $x_{i_{\theta^*}}$ corresponds to the last variable $x_{\pi(u^*)}$ in the sequence whose assignments are different (i.e., where u^* is the largest index u satisfying $w_u \neq w'_u$).

Step 2, Initialize: Recover the $r^{(1)}$-values of layer θ^* using subroutine $\mathcal{D}_2$.

As argued above, the input variable $x_{i_{\theta^*}}$ indeed has its value flip from w to w'. Then, by Claim 6.2, $\mathcal{D}_2$ returns the correct $r^{(1)}$-values $r^{(1)}(\theta^*)$ associated with layer θ^*. Thus, layer θ^* becomes ready.

Algorithm $\mathcal{D}_4$ for recovering $r^{(1)}$-values after a layer is solved

On input 1^n, BP, $y = \hat{f}(w, r)$, $\theta \leq \ell$, $\tilde{w}_{i_{\theta-1}}$, $\tilde{r}^{(1)}(\theta)$, proceed as follows:

For every node t in layer $\theta - 1$, recover $r^{(1)}(t)$ as follows:

1. Find an out-going edge (t, k) whose label is satisfied by $\tilde{w}_{i_{\theta-1}}$. [a]

   [*If $\tilde{w}_{i_{\theta-1}} = w_{i_{\theta-1}}$, then $A(w)_{tk} = 1$. By Property 2 of canonical BPs, k in layer $\theta \leq \ell$ has exactly two incoming edges (t, k) and $(\tilde{t}, k)$ with opposite labels $x_{i_{\theta-1}}$ and $\bar{x}_{i_{\theta-1}}$; thus, $A(w)_{t'k} = 0$ for all $t' \neq t$.*]

2. For every $i < t$, to recover ${r^{(1)}}_{it}$, consider following entry in the randomized polynomial.

$$\begin{aligned} M_{i,k-1}(x) &= \Sigma_{t'k'} \left(R_1 ({r^{(1)}})_{it'} L(x)_{t'k'} R_2 ({r^{(2)}})_{k',k-1} \right) \\ &= \Sigma_{i \leq t'} \left(R_1 ({r^{(1)}})_{it'} \left(A(x)_{t'k} - I_{t'k} \right) \right) \text{ [b]} \\ &= \Sigma_{i \leq t'} \left({r^{(1)}}_{it'} A(x)_{t'k} \right) - {r^{(1)}}_{ik} \end{aligned}$$

   [*If $\tilde{w}_{i_{\theta-1}} = w_{i_{\theta-1}}$, $A(w)_{t' \neq t,k} = 0$ and $A(w)_{tk} = 1$. Thus, $M_{i,k-1}(w) = {r^{(1)}}_{it} - {r^{(1)}}_{ik}$.*]

   Reconstruct the value of $M_{i,k-1}(w)$ from y. Find value $\tilde{r}^{(1)}_{ik}$ in $\tilde{r}^{(1)}(\theta)$, and set $\tilde{r}^{(1)}_{it} = M_{i,k-1}(w) + \tilde{r}^{(1)}_{ik}$.

Finally, output $\tilde{r}^{(1)}(\theta - 1) = \{\tilde{r}^{(1)}_{it}\}_{i<t, t \in V_\theta}$.

[*Remark: When $\tilde{w}_{i_{\theta-1}} = w_{i_{\theta-1}}$, the computation of each $\tilde{r}^{(1)}_{it}$ depends only on value $\tilde{r}^{(1)}_{ik}$, which is different for every pair of (i, t).*]

[a] By Property 1 of canonical BPs, every node not in the last layer has two out-going edges with opposite labels $x_{i_{\theta-1}}$ and $\bar{x}_{i_{\theta-1}}$; thus one of them must be satisfied.

[b] This equality follows from the structures of matrixes R_1, R_2 and L as in Figure 8.

Figure 9: When a layer $\theta - 1$ is solved and layer θ is ready—that is, values $w_{i_{\theta-1}}$ and $r^{(1)}(\theta)$ is known—the $r^{(1)}$-values associated with layer $\theta - 1$ can be recovered from $y = \hat{f}(w, r)$. Then layer $\theta - 1$ becomes ready.

Step 3, Cascade: Starting from that θ^* is ready, "work backwards" from layer $\theta^* - 1$ to the first layer iteratively to recover the values of the input variables associated with these layers. More specifically, in each iteration, starting from that a layer θ is ready (initially $\theta = \theta^*$), recover the assignment $\tilde{w}_{i_{\theta-1}}$ associated with the previous layer $\theta - 1$ using subroutine $\mathcal{D}_3$. Next, use the recovered assignment to recover the $r^{(1)}$-values of layer $\theta - 1$ using subroutine $\mathcal{D}_4$; the algorithm then proceeds to the next iteration with layer $\theta = \theta - 1$, until $\theta = 0$.

In each iteration, when a layer θ is ready, by Claim 6.3, $\mathcal{D}_3$ returns the correct assignment $\tilde{w}_{i_{\theta-1}} = w_{i_{\theta-1}}$ and thus layer $\theta - 1$ becomes solved. Then, combining the fact that layer θ is ready and layer $\theta - 1$ is solved, by Claim 6.4, $\mathcal{D}_4$ returns the correct $r^{(1)}$ values for layer $\theta - 1$, and thus layer $\theta - 1$ becomes ready. Therefore, by a simple induction argument, after i iterations, layers $\theta^* - 1$ to $\theta^* - i$ are solved and layers θ^* to $\theta^* - i$ are ready. At the end of all iterations, values associated with layers $\theta^* - 1$ to 1 are all recovered correctly. In other words, the algorithm computes $w_{\pi(u^*)}, \cdots w_{\pi(1)}$ correctly, which also allows computing $w'_{\pi(u^*)}, \cdots w'_{\pi(1)}$ as $w'_j = w_j \oplus \Delta_j$.

A formal description of the algorithm $\mathcal{D}$ appears in Figure 10. As argued above, relying on Claim 6.1, 6.2, 6.3 and 6.4, the algorithm $\mathcal{D}$ satisfies the requirement of Lemma 6.2, which concludes

the lemma.

Algorithm $\mathcal{D}$ for inverting AIK randomized encodings for Boolean functions

On input 1^n, f^n, $y = \hat{f}(w, r)$, $y' = \hat{f}(w', r)$, proceed as follows:

Generate the canonical branching program BP for computing f^n on n-bit inputs (underlying $\hat{f}$); input variables appear in BP in order $x_{\pi(1)}, \cdots, x_{\pi(n)}$. Then, do:

Estimate difference: Invoke $\mathcal{D}_1(1^n, BP, y, y', i)$ for each $i \in [n]$ to recover the difference between the two assignments. Let $\tilde{D}elta$ be the output, and θ^* be the largest index θ in $[\ell]$ such that layer θ is associated with x_{i_θ} and $\tilde{D}elta_{i_\theta} = 1$. Furthermore, let u^* be the index such that $x_{i_{\theta^*}} = x_{\pi(u^*)}$.

Initialize: Invoke $\mathcal{D}_2(1^n, BP, y, y', \theta^*)$ to get the $r^{(1)}$-values associated with layer θ^*. Let $\tilde{r}^{(1)}(\theta^*)$ be the output. Initialize $\theta = \theta^*$.

Cascade: Proceed in iterations until $\theta = 0$. In each iteration,

- Invoke $\mathcal{D}_3(1^n, BP, y, \theta, \tilde{r}^{(1)}(\theta))$ to recover the assignment to variable $x_{i_{\theta-1}}$ of layer $\theta - 1$; let $\tilde{w}_{i_{\theta-1}}$ be the output.
- Invoke $\mathcal{D}_4(1^n, BP, y, \theta, \tilde{w}_{i_{\theta-1}}, \tilde{r}^{(1)}(\theta))$ to recover the $r^{(1)}$-values associated with layer $\theta - 1$; let $\tilde{r}^{(1)}(\theta - 1)$ be the output.
- Set $\theta = \theta - 1$.

Finally, output $\tilde{w}_{\pi(u^*-1)}, \cdots \tilde{w}_{\pi(1)}$ and $\tilde{w}'_{\pi(u^*-1)}, \cdots \tilde{w}'_{\pi(1)}$ where $\tilde{w}'_j = \tilde{w}_j \oplus \tilde{D}elta_j$.

Figure 10: For every $\mathcal{NC}^1$ Boolean function f, given AIK encodings for two assignments w and w' using shared randomness, partial information of the two assignments can be recovered.

□

6.3.2 Handling Non-Boolean Functions:

Let $f \in \mathcal{NC}^1$ be a function that on input a n bit string, outputs an $l(n)$ bit string. Let f_d be the Boolean function that outputs the d^{th} output bit of f. By construction of the AIK randomized encoding, an encoding $\hat{f}(x; r)$ for n-bit inputs is the concatenation of the encodings $\hat{f}_1(x; r[1]), \cdots, \hat{f}_{l(n)}(x; r[l(n)])$ for each output bit, where $r = r[1], \cdots, r[l(n)]$. Therefore, given encodings y and y' for two different n-bit assignments w and w' using shared randomness r, we can apply algorithm $\mathcal{D}$ on each pair of encodings (y_d, y'_d) w.r.t. Boolean function f_d to recover partial information of w and w'. We show that when w and w' are correlated, the recovered partial information allows to invert $f(w)$ and $f(w')$ with high probability, concluding Theorem 6.2.

Proof Theorem 6.2. We design the algorithm $\mathcal{D}'$ as follows: First parse y and y' as the concatenation of the randomized encoding for each output bit, that is, $y = y_1 \| \cdots \| y_{l(n)}$ and $y = y'_1 \| \cdots \| y'_{l(n)}$, and then invoke $\mathcal{D}$ on $(1^n, f_d, y_d, y'_d)$ for every $d \in [l(n)]$ to recover partial information of w and w'. By Corollary 6.1, in the case where w and w' are correlated, $\mathcal{D}'$ can recover at least $n - \frac{\alpha \log n}{-\log(1-\varepsilon)}$ bits of w and w' with probability $1 - n^{-\alpha}$ for any constant α. Whenever this is true, $\mathcal{D}'$ find an assignment for the remaining $\frac{\alpha \log n}{-\log(1-\varepsilon)}$ bits by brute force, so that the recovered assignments $\tilde{w}$ and $\tilde{w}'$ are pre-images of $f(w)$ and $f(w')$. Therefore, $\mathcal{D}'$ satisfies the statement of the theorem. □

6.4 Inverting AIK Randomized Encoding with Correlated Randomness

The previous section showed that the AIK randomized encoding is easy to invert given instances generated using correlated inputs and *identical* random strings. In this section, we consider instances generated using both correlated input and *correlated* random strings.

We identify certain "regularity" properties, and show that any function $f \in \mathcal{NC}^1$ satisfying these properties—called *well-formed* functions—has the property that their AIK randomized encoding is easy to invert given instances $y = \hat{f}(w, r)$ and $y = \hat{f}(w', r')$ with $(w', r') \xleftarrow{\$} D_\varepsilon(w, r)$. We argue that these properties are not too restrictive by showing that any function $f \in \mathcal{NC}^1$ can be transformed into such another function $f' \in \mathcal{NC}^1$ that is well-formed and "functionally equivalently" to f; here functionally equivalence means that for every input x the output of $f(x)$ can be computed from the output of $f'(x)$ and vice versa. (Furthermore, the transformation can be done efficiently in expected polynomial time.) In other words, the set of functions that are well-formed encompasses all functionalities, and every $\mathcal{NC}^1$ function f can be turned into another function f' that has the "same hardness" as f, whose AIK encodings are easy to invert given instances with correlated inputs and randomness.

6.4.1 Step 1: Invert AIK Randomized Encoding of Well-Formed Functions

Consider $\mathcal{NC}^1$ functions f with the following special properties:

Long output length: On a n-bit input, f outputs $l(n) \geq n\Upsilon$ bits.

High input locality: Every bit of a n-bit input influences at least Υ output bits.

Well diffusion: Every $(1 - \xi)$ fraction of the output bits depends on all n input bits, where $\xi \in \Xi_{f^n} = \{0, 1/l(n), 2/l(n) \cdots, (l(n) - 1)/l(n), 1\}$

We say that f^n is (Υ, ξ)-*well formed*, if f satisfies the above three properties when restricted to n-bit inputs, if for every $n \in \mathbb{N}$, f^n is $(\Upsilon(n), \xi(n))$-well formed for some function $\Upsilon : \mathbb{N} \to \mathbb{N}$, $\xi : \mathbb{N} \to \mathbb{N}$, then we say that f is $(\Upsilon(\cdot), \xi(\cdot))$-*well formed*. We say that a function f is well-formed, if it is $(\Upsilon(\cdot), \xi(\cdot))$-well formed where $\Upsilon(n) = \omega(\log n)$ and $\xi(n) = O(1)$. We show that for every f^n, the above three properties can be efficiently verified and the parameters can be efficiently determined.

Claim 6.5. *There is an efficient algorithm, such that, for every $n \in \mathbb{N}$ and function f, the algorithm on input f^n outputs the* maximal *$\Upsilon \in \mathbb{N}$ and $\xi \in \Xi_{f^n}$ such that f^n is (Υ, ξ)-well formed.*

Proof. The algorithm proceeds as follows: Given f^n that maps n-bit inputs to $l(n)$ bits, to decide Υ, count the number of output bits that are influenced by each input bit and set v to be the minimal number; then set $\Upsilon = \min(v, \lfloor l(n)/n \rfloor)$. To decide ξ, for every input variable x_i, find the set O_i of output bits that *do not* depend on x_i, and let O be the largest set; then, set $1 - \xi = (|O| + 1)/l(n)$. Finally, output (Υ, ξ).

It is easy to see that Υ is the maximal integer w.r.t. which the long output length and high input locality properties hold for f^n. Next, we first show that the well-diffusion property holds w.r.t. ξ and then show that it is also maximal. Assume for contradiction that the well-diffusion property does not hold w.r.t. ξ, that is, there is a set of output bits O' that accounts for a $1 - \xi$ fraction of all output bits (i.e., $|O'| = (1 - \xi)l(n)$), but O' does not depend on all input bits. Let $x_{i'}$ be one of the input bits that O' does not depend on. Then $O' \subseteq O_{i'}$, and $|O_{i'}| \geq |O'| = (1 - \xi)l(n)$. On the other hand, $O_{i'}$ must be smaller than then largest set O, which contains less than $(1 - \xi)l(n)$ number of output bits. This gives a contradiction.

To show that ξ is maximal. Consider another $\xi' \in \Xi_{f^n}$ such that the well-diffusion property holds w.r.t. ξ', that is, every $(1-\xi')$ fraction of the output bits depends on all input bits. We show that $\xi \geq \xi'$. Since every set O_i does not depend on all input bits, they each contains less than $(1-\xi')$ fraction of the output bits. Then $|O| \leq (1-\xi')l(n)-1$ and thus $(1-\xi) = (|O|+1)/l(n) \leq (1-\xi')$. This concludes the claim. □

Theorem 6.3. *For every $\mathcal{NC}^1$ function f, let $\hat{f}$ be the AIK randomized encoding of f using $m(\cdot)$-bit randomness. For every $\varepsilon \in (0, 1/2]$ and $\alpha > 0$, there is a polynomial time algorithm satisfying that for every* well-formed *function f, every sufficiently large $n \in \mathbb{N}$, every $w \in \{0,1\}^n$, and $r \in \{0,1\}^{m(n)}$, the algorithm on input $(1^n, f^n, y = \hat{f}(w,r), y' = \hat{f}(w',r'))$ with $(w',r') \xleftarrow{\$} \mathcal{D}_\varepsilon((w,r))$, inverts $f(w)$ and $f(w')$ with probability $1 - n^{-\alpha}$.*

Proof. Fix any well-formed function f that maps n-bit inputs to $l = l(n)$-bit output and $\varepsilon \in (0, 1/2)$. let f_d be the Boolean function that outputs the d^{th} output bit of f.

Given f^n, by Claim 6.5 one can efficiently find the maximal Υ, ξ such that f^n is (Υ, ξ)-well-formed. Furthermore, for every output bit $d \in [l]$, one can efficiently construct the canonical branching programs BP_d for computing that output bit, with size η_d, adjacency matrix A_d and matrix L_d equal to $A_d - I$ with the first column and last row removed. By the construction of AIK randomized encoding, an encoding $y = \hat{f}(x,r)$ is the concatenation $y = y_1, \cdots, y_l$ of the encoding for each output bit, that is, $y_d = \hat{f}_d(x; r[d])$, where $\hat{f}_d$ is the AIK randomized encoding of the branching program BP_d; each encoding y_d uses independent randomness $r[d]$ and $r = r[1], \cdots, r[l(n)]$. (As before, each $r[d]$ can be parsed into four parts $r[d] = (r[d]^{(1)}, r[d]^{(2)}, r[d]^{(3)}, r[d]^{(4)})$ for different uses when generating the randomized encoding of BP_d.) Note that each output bit d and thus BP_d may not depend on all input variables, but rather a subset of them. Before proceeding to constructing the inverting algorithm, we first consider four subroutines $\mathcal{D}'_1$ to $\mathcal{D}'_4$, similar to that in the proof of Lemma 6.2. All the subroutines works for *arbitrary* w, w' and r, and probabilistically derived r' from r.

Subroutine $\mathcal{D}'_1$ for recovering the differences $\mathcal{D}'_1$ determines whether the i^{th} bits of two assignments w and w' differ or not, given a pair of AIK encoding $y = \hat{f}(w,r), y' = \hat{f}(w',r')$. $\mathcal{D}'_1$ makes use of the subroutine $\mathcal{D}_1$ in the proof of Lemma 6.2 and proceeds as follows: On input $(1^n, f^n, y = \hat{f}(w,r), y' = \hat{f}(w',r'), i)$, for every output bit $d \in [l]$ that depends on x_i, invoke $\mathcal{D}_1(1^n, BP_d, y_d, y'_d, i)$ to obtain $\tilde{D}elta[d]_i$; then output the majority $\tilde{D}elta_i$ of all recovered differences.

We show that the output $\tilde{D}elta_i$ is correct with overwhelming probability.

Claim 6.6. *For every $n \in \mathbb{N}$, $w, w' \in \{0,1\}^n$, and $r \in \{0,1\}^{m(n)}$, $d \in [l(n)]$, in an experiment where r' is sampled from $D_\varepsilon(r)$, the algorithm $\mathcal{D}'_1(1^n, f^n, y = \hat{f}(w,r), y' = \hat{f}(w',r'), i)$ outputs $\tilde{D}elta_i = w_i \oplus w'_i$ with overwhelming probability, over the randomness for sampling r'.*

Proof. We first show that for every output bit d that depends on x_i, the value $\tilde{D}elta[d]_i$ returned by $\mathcal{D}_1$ is correct with probability $1 - \varepsilon$. As described in Figure 6, when $\mathcal{D}_1$ is invoked with input instances y_d, y'_d, it computes its output $\tilde{D}elta[d]_i$ depends only on $r[d]^{(3)}_{tt,k-1,k-1}$ and $r'[d]^{(3)}_{tt,k-1,k-1}$, and the output $\tilde{D}elta[d]_i$ is correct as long as the random coin $r'[d]^{(3)}_{tt,k-1,k-1} = r[d]^{(3)}_{tt,k-1,k-1}$, which happens with probability $1 - \varepsilon$. Thus, $\tilde{D}elta[d]_i = \Delta_i$ with probability $1 - \varepsilon$. Additionally, this probability is independent for different d.

Next, given that f is well-formed, x_i influences $\Upsilon = \Upsilon(n) = \omega(\log n)$ output bits. Each of them is correct with probability $1 - \varepsilon$ independently; therefore, the majority $\tilde{D}elta$ of all $\tilde{D}elta[d]_i$ is correct with overwhelming probability. □

Subroutine $\mathcal{D}'_2$ for recovering $r^{(1)}$ values When an input variable x_i has different value in assignments w and w', subroutine $\mathcal{D}'_2$ is applied to recover the $r^{(1)}$-values associated with layers that depends on x_i. Across the l branching programs $BP_1, \cdots, BP_l$ of f, there could be many layers depending on x_i. However, since the $r^{(1)}$-values for different layers in different BPs are all independently sampled, we recover the $r^{(1)}$-values for each layer θ in each branching program BP_d that depends on x_i separately by invoking $\mathcal{D}'_2(1^n, BP_d, y_d, y'_d, \theta)$, which in turn simply invokes $\mathcal{D}_2$ with the same input. We show that when the instances y_d, y'_d are generated using ε-probabilistically correlated random coins, $\mathcal{D}'_2$ recovers the $r^{(1)}$ values probabilistically.

Claim 6.7. *For every $n \in \mathbb{N}$, $w, w' \in \{0,1\}^n$, $r \in \{0,1\}^{m(n)}$, in an randomized experiment that samples $r' \xleftarrow{\$} D_\varepsilon(r)$, and computes $y = \hat{f}(w, r)$, $y' = \hat{f}(w', r')$, if layer θ in BP_d depends on an variable x_i, and $w_i \neq w'_i$, then $\mathcal{D}'_2(1^n, BP_d, y_d, y'_d, \theta)$ outputs set $\tilde{r}[d]^{(1)}(\theta)$ in which each $\tilde{r}[d]^{(1)}_{it} = r[d]^{(1)}_{it}$ with probability at least $1 - 2\varepsilon$* independently*, over the randomness $r'[d]$ for generating y'_d.*

Proof. $\mathcal{D}'_2$ proceeds identically to $\mathcal{D}_2$. As described in Figure 7, the subroutine $\mathcal{D}_2$ computes each $r^{(1)}$-value $\tilde{r}[d]^{(1)}_{it}$, depending only on $(r[d]^{(3)}_{it,k-1,k-1}, r'[d]^{(3)}_{it,k-1,k-1})$ and $(r[d]^{(1)}_{it}, r'[d]^{(1)}_{it})$ and the output is correct as long as the two pair of random coins are equal, which occurs with probability $1 - 2\varepsilon$. Therefore, each $\tilde{r}[d]^{(1)}_{it}$ output by $\mathcal{D}'_2$ is correct with probability $1 - 2\varepsilon$. Furthermore, since for each i, t, the computation depends on different (independently sampled) random coins, their probabilities of correctness are all independent. □

We say that $\tilde{r}[d]^{(1)}(\theta)$ is a set of **approx-$r^{(1)}$-values** of layer θ in BP_d if it is a *random variable* satisfying the property described in the above claim (i.e., each $r^{(1)}$-value in the set is correct with probability $1 - 2\varepsilon$). When such a random variable is derived, we say that layer θ is approx-ready.

Subroutine $\mathcal{D}'_3$ for recovering assignments In the proof of Lemma 6.2, whenever a layer θ becomes ready, subroutine $\mathcal{D}_3$ can be applied to recover the assignment w_i to the variable x_i associated with the previous layer $\theta - 1$. In this proof, since the recovered $r^{(1)}$-values of a layer may have errors, to compensate this, only when an enough number—$\omega(\log n)$—of layers are approx-ready and all of their previous layers are associated with x_i, can we recover the assignment w_i.

Let $\mathcal{R}_i$ be a set of triplets $(\theta, BP_d, \tilde{r}[d]^{(1)}(\theta))$ satisfying the following properties:

- In each triplet, $\tilde{r}[d]^{(1)}(\theta)$ is a set of approx-$r^{(1)}$-values for layer θ in BP_d.
- Every branching program BP_d appears at most once and their corresponding sets $\{\tilde{r}[d]^{(1)}(\theta)\}$ are all independent.
- The previous layer $\theta - 1$ in BP_d depends on x_i.

The subroutine $\mathcal{D}'_3$ on input $(1^n, f^n, y, i, \mathcal{R}_i)$ proceeds as follows: To find the assignment to x_i, for every triplet $(\theta, BP_d, \tilde{r}[d]^{(1)}(\theta))$ in $\mathcal{R}_i$, invoke $\mathcal{D}_3$ with input $(1^n, BP_d, y_d, \theta, \tilde{r}[d]^{(1)}(\theta))$ and obtain an assignment $\tilde{w}[d]_i$; output the majority $\tilde{w}_i$ of all recovered assignments.

Claim 6.8. *For every $n \in \mathbb{N}$, $w, w' \in \{0,1\}^n$, $r \in \{0,1\}^{m(n)}$, in an experiment that samples $r' \xleftarrow{\$} D_\varepsilon(r)$, computes $y = \hat{f}(w, r)$, $y' = \hat{f}(w', r')$, and produces a set R_i as described above, for every $i \in [n]$, $\mathcal{D}'_3(1^n, f^n, y, i, \mathcal{R}_i)$ outputs w_i with probability at least $1 - \exp(-O(|\mathcal{R}_i|))$.*

Proof. We first show that for every triplet $(\theta, BP_d, \tilde{r}[d]^{(1)}(\theta))$ in $\mathcal{R}_i$, the assignment $\tilde{w}[d]_i$ returned by $\mathcal{D}_3(1^n, BP_d, y_d, \theta, \tilde{r}[d]^{(1)}(\theta))$ is correct with probability $1-2\varepsilon$. This follows since as described in Figure 8, the computation of $\tilde{w}[d]_i$ depends only on $r[d]^{(1)}_{tk} \in r[d]^{(1)}(\theta)$ and the output is correct whenever $\tilde{r}[d]^{(1)}_{tk} = r[d]^{(1)}_{tk}$. By the property of $\mathcal{R}_i$, this occurs with probability $1-2\varepsilon$. Furthermore, since for different triplets, the sets $\{\tilde{r}[d]^{(1)}(\theta)\}$ are independent, the output by $\mathcal{D}_3$ for different triplets are independent. Then, by the Chernoff, the majority of all recovered assignments from $\mathcal{D}_3$ is correct with probability $1-\exp(-O(|\mathcal{R}_i|))$. □

If $\mathcal{R}_i$ contains a super-logarithmic of triplets, the assignment w_i for x_i can be recovered correctly with overwhelming probability. In this case, we say that x_i is solved and all layers (in all branching programs) depending on x_i are solved.

Subroutine $\mathcal{D}'_4$ for recovering $r^{(1)}$-values after a variable x_i is solved After w_i is recovered, for every $\theta-1$ (in some BP_d) that depends on x_i, if its next layer θ (also in BP_d) is approx-ready, then the $r^{(1)}$-values of layer $\theta-1$ can also be recovered probabilistically, and layer $\theta-1$ becomes approx-ready. The subroutine $\mathcal{D}'_4$ on input $(1^n, BP_d, y_d, \theta, w_i, \tilde{r}[d]^{(1)}(\theta))$ simply invokes $\mathcal{D}_4$ with the same input.

Claim 6.9. *For every $n \in \mathbb{N}$, $w, w' \in \{0,1\}^n$, and $r \in \{0,1\}^{m(n)}$, in an experiment that samples $r' \xleftarrow{\$} D_\varepsilon(r)$, computes $y = \hat{f}(w,r)$, $y' = \hat{f}(w',r')$, and produces a set $\tilde{r}[d]^{(1)}(\theta)$ of approx-$r^{(1)}$-values for layer θ in BP_d, $\mathcal{D}_4(1^n, BP_d, y_d, \theta, w_i, \tilde{r}[d]^{(1)}(\theta))$ outputs a set $\tilde{r}[d]^{(1)}(\theta-1)$ of approx-$r^{(1)}$-values for layer $\theta-1$ in BP_d.*

Proof. $\mathcal{D}'_4$ proceeds identically as $\mathcal{D}_4$. As described in Figure 9, the computation of each $r^{(1)}$-value $\tilde{r}[d]^{(1)}_{it} \in \tilde{r}[d]^{(1)}(\theta-1)$ depends on a different value $\tilde{r}[d]^{(1)}_{ik} \in \tilde{r}[d]^{(1)}(\theta-1)$, and whenever $\tilde{r}[d]^{(1)}_{ik}$ is correct (i.e., equals $r[d]^{(1)}_{ik}$), so is $\tilde{r}[d]^{(1)}_{it}$. Since each $\tilde{r}[d]^{(1)}_{ik}$ is correct with probability $1-2\varepsilon$ independently, each $\tilde{r}[d]^{(1)}_{it}$ is correct with probability $1-2\varepsilon$ independently. □

Putting pieces together Given the above four subroutines, the inverting algorithm $\mathcal{D}'$ that on input $(1^n, f^n, y = \hat{f}(w,r), y' = \hat{f}(w',r'))$ proceeds as follows:

Step 1, Preparation: Generate the canonical branching programs $BP_1, \cdots, BP_l$ for each output bit. Compute the maximal Υ, ξ such that f^n is (Υ, ξ)-well formed.

Analysis: By the fact that f is well-formed, f^n is $(\Upsilon'(n), \xi'(n))$-well formed for some $\omega(\log n)$ function Υ' and an $O(1)$ function ξ'. Then by Claim 6.5, $\Upsilon \geq \Upsilon'(n)$ and $\xi' \geq \xi(n)$ are found efficiently.

Step 2, Recover difference: For each i, Compute the difference $\tilde{Delta}_i$ between w_i and w'_i by invoking subroutine $\mathcal{D}'_1$ with input $(1^n, f^n, y, y', i)$. Given $\tilde{Delta}$, for each output bit d, find the *last* layer $\theta[d]$ in BP_d whose associated input variable $x_{i_{\theta[d]}}$ satisfies that $\tilde{Delta}_{i_{\theta[d]}} = 1$. (if all input variables that BP_d depends on have the same values in w and w', set $\theta[d] = \perp$.) Set $\mathcal{F}^* = (\theta[1], \cdots, \theta[l])$.

In the rest of the execution, $\mathcal{D}'$ maintains a length l vector $\mathcal{F}$ where the d^{th} component represents a layer in the d^{th} branching program or is set to $\perp$. We call such a vector a **frontier** and $\mathcal{F}^*$ the initial frontier. We say that a layer θ in BP_d is *above*, *on*, or *below* the frontier $\mathcal{F}$ if $\theta > \theta[d]$, $\theta = \theta[d]$, or $\theta < \theta[d]$. Let U be the set of input variables that appears in

layers above or on the initial frontier, (i.e., $x_i \in U$ if and only if it appears some layer smaller than $\theta[d]$ in some BP_d).

Analysis: By Claim 6.1, $\mathcal{D}_1$ returns the correct difference $\tilde{Delta}_i = \Delta_i = w_i \oplus w'_i$ for each i with overwhelming probability. by the union bound, the whole $\tilde{Delta}$ is correct with overwhelming probability. When this happens, it is indeed the case that every layer above the frontier depends on a variable with the same assignment in w and w', whereas, every layer on the frontier depends on a variable with different assignments. Furthermore, by the fact that w' is derived from w by flipping each bit with probability ε, we show that the number of input variables that appears only above or on the initial frontier is bounded by a logarithmic number with high probability.

Claim 6.10. *For every $\alpha' > 0$, $n \in \mathbb{N}$, $w \in \{0,1\}^n$ and $r \in \{0,1\}^{m(n)}$, in a randomized experiment that samples $w', r' \xleftarrow{\$} \mathcal{D}_\varepsilon(w, r)$, and computes set U as described above, conditioned on $\tilde{Delta} = \Delta$ occurring, the probability that the size of U is greater than $\frac{\alpha' \log n}{-\log(1-\varepsilon)}$ is bounded by $1 - 1/n^{\alpha'}$.*

Proof. Conditioned on $\tilde{Delta} = \Delta$, the initial frontier $\mathcal{F}^*$ contains for every branching program BP_d the last layer that depends on an input variable with different assignments by w and w'. Let U_d and F_d be the set of input variables appearing above and on $\mathcal{F}^*$ in BP_d respectively. Furthermore, let U'_d be the subset of such input variables that does not appear in any $U_{d'}$ with $d' < d$, that is $U'_d = U_d \setminus (U_1 \cup \cdots \cup U_{d-1})$, and similarly, F'_d the set of input variable that does not appear in any $F_{d'}$ with $d' < d$, that is, $F'_d = F_d \setminus (F_1 \cup \cdots, \cup F_{d-1})$. It holds that $|U| = \Sigma_{d\in[l]}(|U'_d| + |F'_d|)$.

Note that (conditioned on $\tilde{Delta} = \Delta$), the sizes of sets U'_d's F'_d's, and U are random variables depending only on the randomness for sampling w' and on the topology of the branching programs. Furthermore, for every possible values $s_1, \cdots, s_l \in \mathbb{N}$ and $\tilde{s}_1, \cdots, \tilde{s}_l \in \mathbb{N}$, the probability that for all d, $|U'_d| = s_d$ and $|F'_d| = \tilde{s}_d$ is

$$\begin{aligned} & \Pr[\forall\ d \in [l],\ |U'_d| = s_d \wedge |F'_d| = \tilde{s}_d] \\ \leq & \prod_{d\in[l]} \left((1-\varepsilon)^{s_d}\varepsilon^{\tilde{s}_d}\right) = \left((1-\varepsilon)^{\Sigma_{d\in[l]}s_d}\right)\left(\varepsilon^{\Sigma_{d\in[l]}\tilde{s}_d}\right) \leq (1-\varepsilon)^{\Sigma_{d\in[l]}(s_d+\tilde{s}_d)} \end{aligned}$$

Then the probability that $|U| \geq \frac{\alpha \log n}{-\log(1-\varepsilon)}$ is bounded by $1/n^\alpha$. This is because for every possible sizes $s_1, \cdots s_l$ and $\tilde{s}_1, \cdots \tilde{s}_l$ of $U'_1 \cdots, U'_l$ and $F'_1 \cdots, F'_l$, such that $|U| = \Sigma_{d\in[l]}(|U'_d| + |F'_d|) = \Sigma_{d\in[l]}(s_d + \tilde{s}_d) \geq \frac{\alpha \log n}{-\log(1-\varepsilon)}$, by the above inequality, the probability they occur is bounded by $1/n^\alpha$. □

The rest of the algorithm is analysed conditioning on that $\tilde{Delta} = \Delta$ and the size of U is bounded.

Step 3, Initialize: For every layer $\theta[d] \neq \perp$ in the initial frontier $\mathcal{F}^*$, invoke subroutine $\mathcal{D}'_2$ with input $(1^n, BP_d, y_d, y'_d, \theta[d])$, and obtain an output set $S[d]$; record triplet $(\theta[d], BP_d, S[d])$ in a set $\mathcal{R}^*$.

Analysis: As argued above, every layer on the frontier depends on an input variable with different assignments. Then, by Claim 6.7, for every possible w, w', r, the set $S[d]$ returned by $\mathcal{D}'_2$ for layer $\theta[d]$ is indeed a set of approx-$r^{(1)}$-values for this layer, and the generation of

$S[d]$ depends only on $r'[d]$. Thus all layers $\theta[d] \neq \perp$ on the initial frontier becomes approx-ready and different sets $\{S[d]\}$ in $\mathcal{R}^*$ are independent.

Step 4, Cascade: Starting from that all layers on the initial frontier are approx-ready, "work backwards" in all the branching programs iteratively to recover the assignment values. More specifically, through out the iterations, the algorithm maintains the a frontier $\mathcal{F} = \{(\theta[d])\}_{d\in[l]}$ and a corresponding set $\mathcal{R} = \{(\theta[d], BP_d, S[d])\}_{\theta[d]\neq\perp\in\mathcal{F}}$ that contains an entry for every non-$\perp$ layer in $\mathcal{F}$, where the set $S[d]$ contains a set of $r^{(1)}$-values for that layer.

Initially $\mathcal{F} = \mathcal{F}^*$ and $\mathcal{R} = \mathcal{R}^*$. In each iteration, starting with a pair $(\mathcal{F}, \mathcal{R})$, set $\mathcal{R}_i$ to the subset of triplets $(\theta[d], BP_d, S[d]) \in \mathcal{R}$ such that layer $\theta[d] - 1$ depends on x_i. If the largest set $\mathcal{R}_{i^*}$ among all $\mathcal{R}_i$'s has more than $\Upsilon\xi/2$ elements, invoke $\mathcal{D}'_3(1^n, f^n, y, i^*, \mathcal{R}_{i^*})$ and record the output assignment $\tilde{w}_{i^*}$. Otherwise, terminate the iterations.

If the iterations are not terminated, prepare the new frontier $\mathcal{F}'$ and $\mathcal{R}'$ for the next iteration as follows: Initialize $\mathcal{F}' = \mathcal{F}$ and $\mathcal{R}' = \mathcal{R}$. For every triplet $(\theta[d], BP_d, S[d]) \in \mathcal{R}_{i^*}$, invoke $\mathcal{D}'_4(1^n, BP_d, \theta[d], \tilde{w}_{i^*}, S[d])$ to obtain an output set $S'[d]$; then, replace this triplet $(\theta[d], BP_d, S[d])$ in $\mathcal{R}'$ with $(\theta[d] - 1, BP_d, S'[d])$ and replace $\theta[d]$ in $\mathcal{F}'$ with $\theta[d] - 1$. Then proceed to the next iteration with the updated $(\mathcal{F}', \mathcal{R}')$.

Analysis: We show that throughout the execution, the pair $\mathcal{F}, \mathcal{R}$ maintained by $\mathcal{D}'$ satisfy the following properties.

- In the frontier $\mathcal{F} = \{(\theta[d])\}_{d\in[l]}$, all non-$\perp$ layers are approx-ready.
- The set $\mathcal{R} = \{(\theta[d], BP_d, S[d])\}_{\theta[d]\neq\perp\in\mathcal{F}}$ contains a set of approx-$r^{(1)}$-values for every non-$\perp$ layer $\theta[d]$ on the frontier $\mathcal{F}$. Moreover, each set $S[d]$ in $\mathcal{R}$ is a random variable depending only on $r'[d]$ and thus are all independent.

By the analysis of step 2, the initial pair $\mathcal{F}^*, \mathcal{R}^*$ satisfies these two property. Next we show that in each iteration, if the pair $(\mathcal{F}, \mathcal{R})$ available at the beginning of the iteration satisfies these two properties, either (1) the iterations terminates, or (2) a correct assignment $\tilde{w}_{i^*} = w_{i^*}$ is recovered and the new pair $\mathcal{F}', \mathcal{R}'$ for the next iteration also satisfy these two properties with overwhelming probability. Then by an induction argument, when the iterations terminates, all the assignments recovered are correct with overwhelming probability.

When an iteration does not terminate, a set $\mathcal{R}_{i^*}$ with more than $\Upsilon\xi/2$ triplets is found, then by Claim 6.8, an invocation to $\mathcal{D}'_3$ returns the valid assignment $\tilde{w}_{i^*} = w_{i^*}$ with probability $1 - \exp(-O(|\mathcal{R}_{i^*}|))$. Since the value $\Upsilon\xi/2$ is super-logarithmic (as $\Upsilon\xi/2 = \Theta(\Upsilon'(n))$), $\tilde{w}_{i^*} = w_{i^*}$ with overwhelming probability.

Conditioned on $\tilde{w}_{i^*} = w_{i^*}$, we show that the new pair $(\mathcal{F}', \mathcal{R}')$ for the next iteration also satisfies the above two properties. Since $(\mathcal{F}', \mathcal{R}')$ only differ from $(\mathcal{F}, \mathcal{R})$ at entries that are contained in $\mathcal{R}_{i^*}$, we only need to analyze the newly updated entries. For each triplet $(\theta[d], BP_d, S[d]) \in \mathcal{R}_{i^*}$, $\theta[d] - 1$ depends on x_{i^*}; thus, layer $\theta[d] - 1$ satisfies that it is solved and its previous layer $\theta[d]$ is approx-ready. Then, by Claim 6.9, for every possible w, w', r, the set $S'[d]$ returned by the invocation of $\mathcal{D}'_4$ (using inputs $\tilde{w}_{i^*} = w_{i^*}$ and set $S[d]$) is indeed a set of approx-$r^{(1)}$-values for layer $\theta[d] - 1$, and the generation of $S'[d]$ depends only on $r'[d]$. Thus, $\theta[d] - 1$ in $\mathcal{F}'$ becomes approx-ready, and $\mathcal{R}'$ contains its corresponding set of approx-$r^{(1)}$-values $S'[d]$ that depends only on $r'[d]$.

Step 5, Output: If the number of input variables x_i whose assignments are not found in $\tilde{w}$ (i.e., $\tilde{w}_i$ is not set) is bounded by $\frac{\alpha' \log n}{-\log(1-\varepsilon)}$, abort and output $\perp$. Otherwise, use brute force to find

assignments tw_i to these variables that together with assignments recovered in Step 4 yield image $f(w)$; set $\tilde{w}' = \tilde{w} \oplus \tilde{D}elta$ and output $\tilde{w}, \tilde{w}'$.

Analysis: By construction, an iteration terminates only if all sets $\mathcal{R}_i$ derived from $\mathcal{R}$ has less than $\Upsilon\xi/2$ triplets. Therefore $\mathcal{R}$ has less than $\Upsilon\xi n/2$ triplets. Since the algorithm maintains the invariant that $\mathcal{R}$ contains a triplet for every non-$\perp$ layer in $\mathcal{F}$, this means at least $(1 - \xi/2)\Upsilon n$ layers in $\mathcal{F}$ are set to $\perp$; let W be the set of corresponding branching programs. By the well-diffusion property of f^n, the branching programs in W depend on all input variables. Furthermore, since Step 4 recovers the assignments for all input variables appearing below the initial frontier $\mathcal{F}^*$ and above the final frontier $\mathcal{F}$, assignments for all input variables appearing below $\mathcal{F}^*$ in branching programs in W are recovered, which contains all variables but these in U. Then conditioned on that the size of U is bounded by $\frac{\alpha' \log n}{-\log(1-\varepsilon)}$. A consistent assignment to variables in U can be recovered using brute force.

Combining the analysis for each step, we have that except with probability $1/n^{\alpha'} + \mu(n)$ for some negligible function μ, $\mathcal{D}'$ outputs value pre-images $\tilde{w}$ and $\tilde{w}'$ for $f(w)$ and $f(w')$. Plugging in $\alpha' = \alpha/2$ concludes the theorem. □

6.4.2 Step 2: From Well-Formed Functions to Any Functions

We show that there is a simple randomized transformation that turns any $\mathcal{NC}^1$ function f into an well-formed function f' that is functionally equivalent in expected polynomial time.

Lemma 6.3. *For every $\mathcal{NC}^1$ function f, and every $n \in \mathbb{N}$, the transformation $\mathsf{Trans}(f^n)$ outputs a function $(f')^n$ that is functionally equivalent to f^n and $(\log^2(\cdot), \xi(\cdot))$-well formed for any constant function $\xi(n) = c$, such that $1 - c - H(c) > 0$, with probability at least $1/2 - \exp(-O(n))$, (where H is the binary entropy of c, $H(c) = -c \log c - (1-c)\log(1-c)$).*

Proof. Consider any $\mathcal{NC}^1$ function f. Towards proving the lemma, we first describe an efficient randomized transformation Trans, that given f^n outputs a function $(f'')^n$, such that, for every *sufficiently large* $n \in \mathbb{N}$, $(f'')^n$ is functionally equivalent to f^n and $(\log^2(n), c)$-well formed for any sufficiently small constant c, with probability close to $1/2$.

Transformation $\mathsf{Trans}(f^n)$: The function f^n maps n bits to $l(n)$ bits. Let $L(n) = \max(l(n), n \log^2 n)$ (that is, $\Upsilon(n) = \log^2 n$). Sample a random $l(n) \times L(n)$ matrix R in $GF(2)$. Construct function $(f'')^n$ that maps an n-bit input x to an $L(n)$-bit output equal to $f^n(x) \times R$ (computed over $GF(2)$). Then output $(f'')^n$ (with R hardwired in).

Note that since $l(n) = \text{poly}(n)$, and f^n can be computed by a circuit of $O(\log n)$ depth, $(f'')^n$ can also be computed by a circuit of $O(\log n)$ depth. We show that for every $n \in \mathbb{N}$, $(f'')^n$ is functionally equivalent to f^n with probability at least $1/2$. The transformation Trans chooses a random matrix R, and construct function $(f'')^n(x) = f^n(x)R$. By construction, $(f'')^n$ is functionally equivalent to f^n whenever R is full rank, which occurs with probability at least $1/2$.

Next, we argue that for every $n \geq 4$, $(f'')^n$ is $(\log^2 n, c)$-well formed for every sufficiently small positive constant c, with all but exponentially small probability. The long output length property follows directly from the construction. It remains to show the high input locality and well diffusion properties. Towards this, we first show that for every output bit d in f^n and every input variable x_i, the probability that the d^{th} output bit depends on x_i is $1/2$. With respect to x_i, every output bit y_j of f^n can be represented as a polynomial over $GF(2)$ with the following form:

$$y_j(x) = x_i P_j^1(x_{\neq i}) + P_j^2(x_{\neq i})$$

where $x_{\neq i}$ represents all but the i^{th} input variables. x_i influences at least one of the output bits of f^n, say it is the $j^{*\text{th}}$ and $P^1_{j^*}(x_{\neq i})$ is a non-zero polynomial. Let $R_{\bullet,d}$ be the d^{th} column of R. The d^{th} output bit z_d of $(f'')^n$ equals to $\langle y, R_{\bullet,d}\rangle$ and can be represented as the following polynomial w.r.t. x_i.

$$z_d(x) = x_i \left(\Sigma_{j\in[l]} R_{jd} P^1_j(x_{\neq i})\right) + \Sigma_{j\in[l]} R_{jd} P^2_j(x_{\neq i})$$

Since over the random choices of $R_{\bullet,d}$, the probability that the polynomial $\Sigma_{j\in[l]} R_{jd} P^1_j(x_{\neq i})$ is non-zero is $1/2$, the probability that the d^{th} output bit of $(f'')^n$ depends on x_i is also $1/2$. Furthermore, this probability is independent for different output bits.

Then, by the Chernoff bound, for any input variable x_i, the probability that it influences less than a quarter of all the output bits is bounded by $\exp(-L(n)/8)$. Thus, for every $n \geq 4$, x_i influences more than $\log^2 n$ output bits with probability $1 - \exp(-L(n)/8)$. Finally, by the union bound, the high input locality property holds with probability $1 - n\exp(-L(n)/8)$. To see the well diffusion property w.r.t. a constant c, consider first any fixed $(1-c)$ fraction of output bits O. For every input variable x_i, the probability that O does not depend on x_i is bounded by $\frac{1}{2^{(1-c)L(n)}}$. Thus by the union bound, the probability that O does not depend on any x_i is bounded by $n/2^{(1-c)L(n)}$. Furthermore, since there are at most $2^{(H(c)L(n))}$ different possible sets O. By the union bound again, the probability that there is a set of $(1-c)$ fraction of output bits that does not depend on all input variables is bounded by $n/2^{(1-c-H(c))L(n)}$. When c is sufficiently small, such that, $1 - c - 2H(c) > 0$, this probability is exponentially small.

Overall, by the union bound, for every $n \geq 4$, the transformation Trans outputs a function $(f'')^n$ that is functionally equivalent to f^n and is $(\log^2(n), c)$-well-formed for all sufficiently small constant c such that $1 - c - 2H(c) > 0$, with probability

$$p(n) = \frac{1}{2} - \frac{n}{e^{(L(n)/8)}} - \frac{n}{2^{(1-c-H(c))L(n)}} \geq \frac{1}{2} - \frac{n}{e^{(n/8)}} - \frac{n}{2^{(1-c-H(c))n}}$$

Note that this probability is independent of the function f, and there exists a constant $N \in \mathbb{N}$ such that for every $n \geq N$, the above probability is greater than $1/4$. Using this N, we construct another transformation Trans' that outputs a function $(f')^n$ that is functionally equivalent for all inputs (rather than only for sufficiently long inputs).

Fix any constant c such that $1 - c - 2H(c) > 0$. The new transformation Trans' on input a function f^n, simply outputs $(f')^n = f^n$ if $n < N$; otherwise, if $n \geq N$, it repeated invokes $\mathsf{Trans}(f^n)$ until the output $(f'')^n$ is indeed functionally equivalent to f^n and $(\log^2 n, c)$-well formed, and outputs $(f')^n = (f'')^n$. (The functionally equivalence property can be efficiently checked by checking whether the random matrix R that Trans chooses is full rank.) Since for every $n \geq N$, the probability that the function output by Trans satisfies these two properties is larger than $1/4$, Trans' runs in expected polynomial time. Furthermore, consider the function $f' = \{(f')^n\}_{n\in\mathbb{N}}$ output by Trans'; f' is functionally equivalent to f for all inputs, and is $(\log^2(\cdot), \xi(\cdot))$-well-formed for $\xi(n) = c$. This concludes the lemma.

□

Lemma 6.3 implies that for every function $f \in \mathcal{NC}^1$, there exists another function $f' = \{(f')^n\}_{n\in\mathbb{N}} \in \mathcal{NC}^1$, such that, f' is well-formed and functionally equivalent to f. Therefore, f' has the same hardness as f. Since f' is well-formed, by Theorem 6.3, $\hat{f}'$ is easy to invert given instances with correlated inputs and randomness. Therefore, we obtain the following Corollary 6.2.

Corollary 6.2. *For every function f in $\mathcal{NC}^1$, there exists another function $f' \in \mathcal{NC}^1$ that is functionally equivalent to f and well-formed. Furthermore, the AIK randomized encoding $\hat{f}'$ of*

f' uses $m(\cdot)$-bit random coins, and for every $\varepsilon \in (0, 1/2]$ and $\alpha > 0$, there is a polynomial time algorithm satisfying that for every sufficiently large $n \in \mathbb{N}$, every $w \in \{0,1\}^n$ and $r \in \{0,1\}^{m(n)}$, the algorithm on input $(1^n, f^n, (\hat{f}')^n, y = \hat{f}'(w,r), y' = \hat{f}'(w',r'))$ with $(w',r') \stackrel{\$}{\leftarrow} \mathcal{D}_\varepsilon((w,r))$, inverts $f(w)$ and $f(w')$ with probability $1 - n^{-\alpha}$.

References

[ABW10] Benny Applebaum, Boaz Barak, and Avi Wigderson. Public-key cryptography from different assumptions. In *STOC*, pages 171–180, 2010.

[ACM+13] Per Austrin, Kai-Min Chung, Mohammad Mahmoody, Rafael Pass, and Karn Seth. On the (im)possibility of tamper-resilient cryptography: Using fourier analysis in computer viruses. *IACR Cryptology ePrint Archive*, 2013:194, 2013.

[AHI11] Benny Applebaum, Danny Harnik, and Yuval Ishai. Semantic security under related-key attacks and applications. In *ICS*, pages 45–60, 2011.

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in nc^0. In *FOCS*, pages 166–175, 2004.

[Ale03] Michael Alekhnovich. More on average case vs approximation complexity. In *FOCS*, pages 298–307, 2003.

[ALM+98] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and intractability of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.

[App11] Benny Applebaum. Randomly encoding functions: A new cryptographic paradigm (survey). The 5th International Conference on Information Theoretic Security, 2011.

[App12] Benny Applebaum. Pseudorandom generators with long stretch and low locality from random local one-way functions. In *STOC*, pages 805–816, 2012.

[AS98] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998.

[BAB93] Ishai Ben-Aroya and Eli Biham. Differtial cryptanalysis of lucifer. In *CRYPTO*, pages 187–199, 1993.

[BC10] Mihir Bellare and David Cash. Pseudorandom functions and permutations provably secure against related-key attacks. In *CRYPTO*, pages 666–684, 2010.

[BCM11] Mihir Bellare, David Cash, and Rachel Miller. Cryptography secure against related-key attacks and tampering. In *ASIACRYPT*, pages 486–503, 2011.

[BFKL93] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In *CRYPTO*, pages 278–291, 1993.

[Bih94] Eli Biham. New types of cryptanalytic attacks using related keys. *J. Cryptology*, 7(4):229–246, 1994.

[BK03] Mihir Bellare and Tadayoshi Kohno. A theoretical treatment of related-key attacks: Rka-prps, rka-prfs, and applications. In *EUROCRYPT*, pages 491–506, 2003.

[BKS13] Boaz Barak, Guy Kindler, and David Steurer. On the optimality of semidefinite relaxations for average-case and generalized constraint satisfaction. In *ITCS*, pages 197–214, 2013.

[BMOS05] Nader H. Bshouty, Elchanan Mossel, Ryan O'Donnell, and Rocco A. Servedio. Learning dnf from random walks. *J. Comput. Syst. Sci.*, 71(3):250–265, 2005.

[BQ12] Andrej Bogdanov and Youming Qiao. On the security of goldreich's one-way function. *Computational Complexity*, 21(1):83–127, 2012.

[BS90] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. In *CRYPTO*, pages 2–21, 1990.

[BS91a] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. *J. Cryptology*, 4(1):3–72, 1991.

[BS91b] Eli Biham and Adi Shamir. Differential cryptanalysis of snefru, khafre, redoc-ii, loki and lucifer. In *CRYPTO*, pages 156–171, 1991.

[BS91c] Eli Biham and Adi Shamir. Differential cryptoanalysis of feal and n-hash. In *EUROCRYPT*, pages 1–16, 1991.

[BS92] Eli Biham and Adi Shamir. Differential cryptanalysis of the full 16-round des. In *CRYPTO*, pages 487–496, 1992.

[CEMT09] James Cook, Omid Etesami, Rachel Miller, and Luca Trevisan. Goldreich's one-way function candidate and myopic backtracking algorithms. In *TCC*, pages 521–538, 2009.

[DGK+10] Yevgeniy Dodis, Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Public-key encryption schemes with auxiliary inputs. In *TCC*, pages 361–381, 2010.

[DKL09] Yevgeniy Dodis, Yael Tauman Kalai, and Shachar Lovett. On cryptography with auxiliary input. In *STOC*, pages 621–630, 2009.

[DLSS13] Amit Daniely, Nati Linial, and Shai Shalev-Shwartz. More data speeds up training time in learning halfspaces over sparse vectors. In *NIPS*, pages 145–153, 2013.

[DOPS04] Yevgeniy Dodis, Shien Jin Ong, Manoj Prabhakaran, and Amit Sahai. On the (im)possibility of cryptography with imperfect randomness. In *FOCS*, pages 196–205, 2004.

[Fei02] Uriel Feige. Relations between average case complexity and approximation complexity. In *STOC*, pages 534–543, 2002.

[FK01] Uriel Feige and Joe Kilian. Heuristics for semirandom graph problems. *J. Comput. Syst. Sci.*, 63(4):639–671, 2001.

[Fla08] Abraham Flaxman. A spectral technique for random satisfiable 3cnf formulas. *Random Struct. Algorithms*, 32(4):519–534, 2008.

[GGR96] Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. In *FOCS*, pages 339–348, 1996.

[GL10] David Goldenberg and Moses Liskov. On related-secret pseudorandomness. In *TCC*, pages 255–272, 2010.

[Gol00] Oded Goldreich. Candidate one-way functions based on expander graphs. *IACR Cryptology ePrint Archive*, 2000:63, 2000.

[Hås01] Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.

[HDWH12] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In *Proceedings of the 21st USENIX Security Symposium*, August 2012.

[IK02] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, pages 244–256, 2002.

[IP01] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

[Knu92] Lars R. Knudsen. Cryptanalysis of loki91. In *AUSCRYPT*, pages 196–208, 1992.

[McE78] Robert J McEliece. A publickey system based on algebraic coding theory pages. *DSN Progress Report, Jet Propulsion lab*, 44:114–116, 1978.

[OW12] Ryan O'Donnell and David Witmer. Goldreich's prg: Evidence for near-optimal polynomial stretch. Manuscript, 2012.

[PW11] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. *SIAM J. Comput.*, 40(6):1803–1844, 2011.

[RS10] Alon Rosen and Gil Segev. Chosen-ciphertext security via correlated products. *SIAM J. Comput.*, 39(7):3058–3088, 2010.

[ST03] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis (motivation and discrete models). In *WADS*, pages 256–270, 2003.

[ST04] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. ACM*, 51(3):385–463, 2004.

[Val84] Leslie G. Valiant. A theory of the learnable. In *STOC*, pages 436–445, 1984.

[Wee12] Hoeteck Wee. Public key encryption against related key attacks. In *Public Key Cryptography*, pages 262–279, 2012.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de