Electronic Colloquium on Computational Complexity, Report No. 175 (2012)

On the One-Way Function Candidate Proposed by Goldreich*

James Cook[1] **, Omid Etesami[2] ***, Rachel Miller[3] †, and Luca Trevisan[4] ‡

[1] U.C. Berkeley, Computer Science Division
jcook@cs.berkeley.edu
[2] EPFL, Laboratoire d'Algorithmique
omid.etesami@epfl.ch
[3] MIT, Computer Science and Artificial Intelligence Laboratory
r_miller@mit.edu
[4] Stanford, Computer Science Department
trevisan@stanford.edu

Abstract. A function f mapping n-bit strings to m-bit strings can be constructed from a bipartite graph with n vertices on the left and m vertices on the right having right-degree d together with a predicate $P : \{0,1\}^d \to \{0,1\}$. The vertices on the left correspond to the bits of the input to the function and the vertices on the right correspond to the bits of the output. The value of each output bit is computed by evaluating the predicate over the input bits corresponding to its neighbors. Goldreich (ECCC 2000) conjectured that this function f is one-way for $m = n$ and $d = \Theta(1)$ or $d = \Theta(\log n)$, when the vertices on the right "expand" and the predicate P is a random non-linear predicate.

Inverting f as a one-way function by definition means finding an element in the preimage $f^{-1}(f(x))$ of the image of a random input x under the function f. We bound the expected size of this preimage when the bipartite graph is a random bipartite graph with right-degree d. Our result is for when the predicate P is a typical random predicate or $P(x_1, \ldots, x_d) = x_1 \oplus \ldots \oplus x_{d-h} \oplus Q(x_{d-h+1}, \ldots, x_d)$ where Q is an arbitrary predicate having at most $d - \Theta(d)$ variables.

Inverting the function can be seen as a constraint satisfaction problem with a "planted solution". One can use backtracking algorithms to find this solution. Using the bound on the size of the preimage, we prove those backtracking algorithms that are "myopic" and those backtracking algorithms that are "drunk" cannot invert the function in better than exponential time on average.

Myopic backtracking algorithms are ones in which during the first levels of the backtracking tree, the algorithm has a limited view of the image $f(x)$ for which the algorithm wants to find an element in the preimage $f^{-1}(f(x))$. Our proof for myopic backtracking algorithms builds upon the work of Alekhnovich,

* Some of the results in this paper appeared previously in the 6th Theory of Cryptography Conference in 2009 [8].

** This material is based upon work supported by the National Science Foundation under Grant Nos. CCF 1017403 and CCF 0729137, and by the National Sciences and Engineering Research Council of Canada under a PGS award.

*** This material is based upon work in UC Berkeley supported by the National Science Foundation under Grant Nos. CCF 1017403 and CCF 0729137, and work in EPFL supported by European Research Council Grant No. 228021-ECCSciEng.

† Work done at U.C. Berkeley, supported by an NSF SUPERB fellowship, and at MIT, supported by an NDSEG fellowship.

‡ This material is based upon work supported by the National Science Foundation under Grant Nos. CCF 1017403 and CCF 0729137, and by the BSF under grant 2002246.

ISSN 1433-8092

Hirsch, and Itsykson (2005) where they instead consider solving a system of linear equations each with three variables, that is, when $P = x_1 \oplus x_2 \oplus x_3$.

Drunk backtracking algorithms for a constraint satisfaction problem are ones in which at each point in the backtracking tree, even though the algorithm can choose any variable to fix next, the bit that is first tried for that variable has to be random. Our proof for drunk backtracking algorithms is similar to those of Itsykson (CSR 2010) and Miller (thesis 2009).

We show that these lower bounds also hold when the backtracking algorithm is allowed to eliminate "pure literals" and "unit clauses" as in DPLL algorithms.

Since both being myopic and being drunk are merely theoretical restrictions that allowed us to prove lower bounds, we also performed an experimental analysis by running one of the best available SAT solvers on the SAT instance equivalent to the constraint satisfaction problem corresponding to inverting the function. The solver seemed to take exponential running time.

1 Introduction

Goldreich [12] proposed in 2000 a candidate one-way function construction based on expanding graphs. His construction is parameterized by the choice of a bipartite graph with n vertices per side and right-degree d (where d is either a constant independent of n, or grows very moderately as $O(\log n)$) and of a boolean predicate $P : \{0,1\}^d \to \{0,1\}$. To compute the function, on input $x \in \{0,1\}^n$ we label the vertices on the left by the bits of x, and we label each vertex on the right by the value of P applied to the labels of the neighbors. The output of the function is the sequence of n labels of the vertices on the right.

Goldreich's Function and Cryptography in $\mathbf{NC^0}$. A function is computable in NC^0 if every bit of the output depends only on a constant number of bits of the input. One can view any NC^0-computable function as a generalization of Goldreich's function in which different predicates can be used for different bits of the output, and in which the graph is allowed to be arbitrary, subject to having bounded right-degree.

Cryan and Miltersen [9] first raised the question of whether cryptographic primitives can be computed in NC^0 in a work focused on pseudorandom generators. Mossel, Shpilka and Trevisan [18] construct, for arbitrarily large constant c, a function $f : \{0,1\}^n \to \{0,1\}^{cn}$ based on a bipartite graph of right-degree 5 and the fixed predicate $P(x_1, \cdots, x_5) := x_1 \oplus x_2 \oplus x_3 \oplus (x_4 \wedge x_5)$, and show that the function computes a small-bias generator. Such a construction may in fact be a pseudorandom generator, and hence a one-way function.[5]

Applebaum, Ishai and Kushilevtiz [4,5] show that, under standard assumptions, one can construct one-way functions and pseudorandom generators that can be computed in NC^0; their one-way functions are computable with right-degree 3.[6] In their construction, the graph encodes the computation of a log-space machine computing a one-way function (not in NC^0) that is used as a primitive.

Applebaum, Barak and Wigderson [3] build a public-key cryptosystem using assumptions that include that Goldreich's function with predicates of the form $MAJ(x_1 \oplus \cdots \oplus x_{d/3}, x_{d/3+1} \oplus \cdots \oplus x_{2d/3}, x_{2d/3+1} \oplus \cdots \oplus x_d)$ is a pseudorandom generator; this assumption is stronger than Goldreich's original conjecture.

Applebaum [2] constructs pseudorandom generators based on the assumption that Goldreich's function with a random predicate and graph is one-way. He constructs a linear-stretch pseudorandom generator in NC^0, a polynomial-stretch pseudorandom generator when the degree of each output bit is $\omega(1)$, and a polynomial-stretch generator in NC^0 with inverse polynomial distinguishing advantage.

In this paper, we are interested in the security of Goldreich's original proposal as a one-way function, implemented using a random graph and a predicate that either has a certain form or is chosen randomly.

Goldreich's Function and DPLL Algorithms. Inverting Goldreich's one-way function (and, indeed, inverting any one-way function that is computable in NC^0) can be seen as the task of finding a solution to a constraint satisfaction problem with a planted solution. A plausible line of attack against such functions is to employ a general-purpose SAT solver to solve the corresponding constraint satisfaction problem. We performed an experimental study using MiniSat, which has been used to solve instances with several thousand variables,

[5] The graph used in this construction, however, is not a random graph or a strong expander graph of right-degree 5, so this is not an instantiation of Goldreich's proposal.

[6] This is the best possible, because it is easy to show that no function based on a bipartite graph of right-degree 2 can be one-way, by reducing the problem of finding the inverse to a 2SAT instance.

and is generally agreed to be one of the best publicly available SAT solvers. Using a random graph of right-degree 5, and the predicate $(x_1 \oplus x_2 \oplus x_3 \oplus (x_4 \wedge x_5))$, we observed an exponential increase of the running time as a function of the input length; an attack with MiniSat appears infeasible already for very modest input lengths (a few hundred bits). See Section 8.

Our goal in this paper is to provide a rigorous justification for these experimental results, and to show that "DPLL-style" algorithms based on backtracking (such as most general SAT solvers) cannot break Goldreich's construction in sub-exponential time. We restrict our work to algorithms that instantiate variables one at a time in an order chosen adaptively by a "scheduler" procedure. When instantiating a variable, the scheduler also decides whether to recurse on the instance obtained by fixing the variable to zero and then to the instance obtained by fixing the variable to one, or vice versa. A recursive branch stops if the current partial assignment contradicts one of the constraints in the instance. The program terminates once it finds a satisfying assignment.

When such an algorithm runs on an unsatisfiable instance, the transcript of the algorithm's substitutions gives a "tree-like resolution proof" of unsatisfiability. A number of techniques prove exponential lower bounds on the size of tree-like resolutions proofs of unsatisfiability, and therefore imply lower bounds for the running time of any such algorithm on such instances, regardless of how the scheduler is designed.

When dealing with *satisfiable* instances, however, one cannot prove non-trivial lower bounds without putting some restriction on the scheduler. If unrestricted in computational power, the scheduler could simply compute a satisfying assignment and assign the variables accordingly, giving an algorithm that converges in a linear number of steps.

The Lower Bound of Alekhnovich et al. Alekhnovich, Hirsch and Itsykson [1] consider two such restrictions: they consider (i) "myopic" algorithms in which the scheduler chooses the next assignment to make based on only a bounded number of clauses of the current formula, and (ii) "drunk" algorithms in which the order of variables is chosen arbitrarily by the scheduler, but the choice of whether to first assign zero or one to the chosen variables is made randomly with equal probability. Their results for drunk algorithms were proven for carefully designed instances unrelated to Goldreich's function. Alekhnovich et al. proved lower bounds against myopic algorithms for a fixed predicate through a reduction to the problem of certifying unsatisfiability: they show that after a myopic algorithm assigns a certain number of variables, with high probability it is left with an instance that is unsatisfiable, but which has no sub-exponential size tree-like resolution proof of unsatisfiability. Hence, with high probability the algorithm must take an exponential amount of time to discover it has chosen a bad partial assignment.

Our Results The result of Alekhnovich et al. applies to myopic algorithms for random instances of 3XOR with a planted solution, giving a lower bound for myopic DPLL inversion algorithms for the instantiation of Goldreich's proposal using the 3XOR predicate.

Unfortunately, despite its resistance to myopic algorithms, Goldreich's construction instantiated with the 3XOR predicate is easily inverted via Gaussian elimination. Furthermore, their result for drunken algorithms does not seem to relate to Goldreich's function. We extend the result of Alekhnovich et al. to use either a random predicate (as Goldreich originally proposed) or predicates of the form $(x_1 \oplus \cdots \oplus x_{d-h} \oplus Q(x_{d-h+1}, \ldots, x_d))$ (inspired by the work of Mossell et al). We also show that the framework we build to analyze myopic algorithms can be adapted to work for drunk algorithms as well, to produce a result similar to those of Itsykson [14] and

Miller [17]. In order to extend the work of Alekhnovich et al. to the setting of Goldreich's one way function, we make the following contributions:

- The proof in [1] assumes that there is a unique solution, and this is not true in our setting. We show that the proof carries over if one assumes that the total number of pairs x, y such that $f(x) = f(y)$ is at most $2^{(1+\epsilon)n}$ for small ϵ. We are able to show an upper bound that such a condition is satisfied by the predicates we consider and by a random choice of graph, with $\epsilon = 2^{-\Omega(d)}$. This upper bound, proved in Section 3, is interesting in itself: It is an upper bound on the expected size of preimages, or equivalently many-to-oneness, of Goldreich's function. Panjwani [19] has also experimentally analyzed the size of pre-images of Goldreich's function.
- The proof in [1] uses the linearity of the constraints. We show that it is sufficient for the predicate to have nearly balanced output even after many variables have been fixed to arbitrary values. For example, a d-ary parity remains perfectly balanced even after $d-1$ variables are assigned arbitrary variables. The predicate $(x_1 \oplus \cdots \oplus x_{d-2} \oplus (x_{d-1} \wedge x_d))$ remains perfectly balanced even after $d-3$ variables are assigned arbitrary variables, and a random predicate remains ϵ-close to balanced after any $d - O(\log d/\epsilon)$ variables are fixed to arbitrary values. (These parameters are sufficient for our proof to go through.)
- The proof in [1] uses the fact that all constraints have arity three, and relies on degree-three strongly expanding bipartite graphs. Furthermore, it requires the adjacency matrix of the graph to have full rank, so that the solution will be unique. There is nothing unique about arity three; as mentioned above, we adapt their methods to work for predicates of larger arity. We also remove the requirement for the adjacency matrix to have full rank; instead, we use random graphs, for which we prove that Goldreich's function is not far from one-to-one. However, there is a non-negligible chance that a random graph will violate the strong expansion condition required by the approach in [1], so in Sections 5.3 and 6.4 we show that the lower bound still holds for graphs with a weaker form of expansion, which is violated with only negligible probability.
- DPLL algorithms are allowed to apply two special rules to simplify their input, called unit clause elimination and pure literal elimination, which makes it more difficult to prove lower bounds for DPLL algorithms. However, in the case of linear predicates considered by [1], these two new rules add little to the power of the backtracking algorithm. In considering non-linear predicates, we do further work in Section 7 to show that lower bounds for backtracking algorithms without these two rules imply lower bounds for DPLL algorithms.

With these contributions, we are able to show an exponential lower bound for both myopic and drunk DPLL algorithms in a construction that uses a random graph and a predicate which is either random or of the form $(x_1 \oplus \cdots \oplus x_{d-h} \oplus Q(x_{d-h+1}, \ldots, x_d))$.

Goldreich's Analysis Goldreich [12] considered the following algorithm (as an obvious first attack) for computing x given $y = f(x)$. The algorithm proceeds in n steps, revealing the output bits one at a time. Let R_i be the set of inputs connected to the first i outputs. Then in the ith step, the algorithm computes the list L_i of all strings in $\{0,1\}^{R_i}$ which are consistent with the first i bits of y. The final list L_n enumerates the set $f^{-1}(y)$. Goldreich proves that if the graph satisfies an expansion condition, the expected size of one of the sets L_i is exponentially large, and so this inversion algorithm runs in time exponential in the input length. Panjwani [19] experimentally verified this result.

The above algorithm is a weaker version of a myopic backtracking algorithm where before assigning values to the bits in R_i, the algorithm has no knowledge of y except its first i bits.[7] For this reason, our lower bounds for myopic algorithms are more general.

Proof Overview of Upper Bound on Expected Size of Preimages To calculate the expected size of the preimage, we note that it is equivalent to calculating the probability that the image of two different n-bit inputs collide under Goldreich's function. When the graph used for Goldreich's function is random, this probability can be calculated in terms of the probability that the predicate P outputs the same bit when it receives two d-bit inputs when the bits of these two inputs are random but bitwise correlated.

We need to show that a random predicate P performs well for all the infinite number of ways in which the two d-bit inputs to P can be bitwise correlated. Since the number of ways is infinite, we cannot directly apply a union bound. Rather, we apply the union bound after showing how to "summarize" performance for all the ways of bitwise correlation by performance on a finite number of other types of correlations.

Proof Overview of Lower Bound for Backtracking Algorithms Our proof is similar to the proof of Alekhnovich et al. [1] that myopic backtracking algorithms take an exponentially long time to solve systems of linear equations.

Let $\mathcal{A}$ be a myopic or drunk backtracking algorithm. We pick a random $x \in \{0,1\}^n$, and run $\mathcal{A}$ on input $b = f(x)$. Our goal is to show that $\mathcal{A}$ will run for a long time before returning any $x' \in f^{-1}(b)$, but we begin with an easier goal: to show that $\mathcal{A}$ will either run for a long time or return a value that is not *exactly* equal to x. In the case that f is an injective function, these goals are one and the same. In the general case, Lemma 3.1 allows us to reduce the harder goal to the easier goal.

Our strategy to prove the easier goal is to show first that with high probability $\mathcal{A}$ will choose an incorrect value for a variable, and second, that it will take a long time to recover from its mistake. We are only able to prove the second part when the mistake the algorithm made is "locally consistent" (Definition 4.4).

At every point during the execution of $\mathcal{A}$, its partial truth assignment ρ can be in one of three states:

1. ρ is consistent with x.
2. ρ is not consistent with x, but is locally consistent.
3. ρ is not locally consistent.

We need to show that the algorithm reaches state 2 with high probability.

In order to make our task simpler, we start in by modifying $\mathcal{A}$ so that it becomes a *clever* algorithm that never enters state 3 before it makes a large number of assignments (Lemma 4.13). Then we show (in Lemma 5.1 when $\mathcal{A}$ is myopic and in Lemma 6.4 when $\mathcal{A}$ is drunk) that with high probability, the algorithm reaches state 2.

[7] We introduce myopic backtracking algorithms in Definitions 2.8 and 2.9. The list L_i mentioned above corresponds to the nodes at level $|R_i|$ of the backtracking tree of the myopic backtracking algorithm. To simulate the algorithm considered by Goldreich by a myopic backtracking algorithm, the scheduler of the backtracking algorithm does not need to look at the output to decide which variable to assign next, nor to decide which value to try assigning to that variable first.

History of This Work A subset of the results in this paper appeared in the 6th Theory of Cryptography Conference in 2009 [8]. Specifically, we presented our lower bound for myopic backtracking algorithms on instances of Goldreich's function that use a random graph that satisfies an expansion condition, and the predicate $x_1 \oplus \ldots \oplus x_{d-2} \oplus (x_{d-1} \wedge x_d)$. Though Goldreich [12] suggested using random predicates, our proof was missing a bound on the size of pre-images of Goldreich's function for this case.

Itsykson [14] later proved a similar lower bound for drunk DPLL algorithms for random graphs and the more general predicate $x_1 \oplus \ldots \oplus x_{d-h} \oplus Q(x_{d-h+1}, \ldots, x_d)$. He also noted that our proof applies to this more general predicate. Miller [17] independently looked at lower bounds for drunk algorithms inverting Goldreich's function.

This version of our work differs from the conference version by the following additions:

– We prove our result for random predicates by bounding the size of pre-images of Goldreich's function. In Section 3, we describe a property that is both computable and sufficient for the needs of our proof, and which a uniformly random d-ary predicate satisfies with high probability[8]. This brings our work more in line with Goldreich's original suggestion.
– We show that our lower bounds apply to DPLL algorithms, which are backtracking algorithms with the additional ability to apply certain simplification rules. See Section 7.
– Our original expansion requirement (as well as those of [14] and [17]) is violated by a constant-degree uniformly random graph with polynomial probability. We weaken our expansion condition by allowing small non-expanding sets. The probability that a random graph violates our weaker condition is exponentially small. See Section 5.2.
– We include our own proof of the lower bound for drunk algorithms, which is similar to the work of Itsykson and Miller. The proofs of the myopic and drunk lower bounds given in this paper follow each other very closely, diverging for only a few steps.

Related Work on Inverting Goldreich's Function Besides Itsykson's independent lower bound on drunk DPLL backtracking algorithms mentioned before, there has been other further work since the publication of the conference version of this work.

Bogdanov and Qiao [7] present an efficient algorithm for inverting instances of Goldreich's function where the number of output bits is much larger than the number of input bits, and the predicate is correlated with one or two of its inputs. Notice that Goldreich's original proposal suggests that the number of input bits be equal to the number of output bits.

Itsykson and Sokolov [13] find a lower bound for the running time of myopic and drunk DPLL algorithms for an explicit construction of Goldreich's function whose pre-images have size $2^{o(n)}$, which is interesting compared to our results for random graphs. However, their function partitions the input variables into two parts: $o(n)$ variables which affect the output non-linearly, and $n - o(n)$ variables which affect the output linearly. The fact that the non-linear part is small allows an algorithm to invert the function in $2^{o(n)}$ time.

In a more recent work, Itsykson and Sokolov [15] find a lower bound for myopic DPLL algorithms with the addition of a myopic *cut heuristic* which allows the algorithm to remove branches from the backtracking tree. This is significant since previous work, as well as this paper, rely on the fact that a DPLL algorithm

[8] More exactly, the probability is $1 - 2^{-2^{\Omega(d)}}$ by Lemma 3.8.

would encounter a large subtree containing no solutions, and would be forced to explore the entire subtree before continuing. The hard formulas they consider are systems of linear equations, which are equivalent to instances of Goldreich's function with an XOR predicate.

Open Questions Our work adds motivation for further experimental and rigorous analysis of Goldreich's construction.

The limitation of the present work is the somewhat artificial nature of both myopic algorithms and drunk algorithms. Myopic algorithms fail to capture certain natural "global" heuristics used in SAT solvers. Since the algorithm is required to work only with partial information on the object given as an input, negative results for myopic algorithms are similar in spirit (but very different technically) to results on "space bounded cryptography." Drunk algorithms are restricted in a way that is more computational than information-theoretic, but the random selection of variable values is clearly contrived. Despite these limitations, we hope our results will serve as an important step toward lower bounds for more general classes of algorithms.

Another interesting goal would be to show that no "variation of Gaussian elimination" can invert Goldreich's function when non-linear predicates are used. Unfortunately, it is not clear how to even formalize such a statement. For example, here is an algorithm for inverting Goldreich's function with predicates of the form $P(x) = x_1 \oplus \ldots \oplus x_{d-2} \oplus (x_{d-1} \wedge x_d))$ which uses a combination of backtracking and Gaussian elimination: first, consider a graph H on n vertices, where for every constraint of the form $x_{i_1} \oplus \ldots \oplus x_{i_{d-2}} \oplus (x_{i_{d-1}} \wedge x_{i_d}) = b_j$ there is an edge between the i_{d-1}-th and i_d-th vertices. Find a vertex cover for the random graph H, and backtrack to set the values of the variables corresponding to this vertex cover. Once these variables have been set, the values of the remaining variables can be found using Gaussian elimination. This will give an improved running time (though still exponential) because the vertex cover contains a strict subset of the n variables.

2 Definitions and Statement of Theorems

2.1 The Problem of Inverting Goldreich's Function

Definition 2.1 *The* **one-way function candidate** $f = f_{P,G} : \{0,1\}^n \to \{0,1\}^m$ *proposed by* **Goldreich** *is parameterized by*

- *a d-ary predicate $P : \{0,1\}^d \to \{0,1\}$, and*
- *a bipartite graph G with n vertices on the left and m vertices on the right having right-degree d.*

We represent the bipartite graph as $G \in [n]^{m \times d}$ where $[n] = \{1, \ldots, n\}$. In this representation, the set of vertices on the left is $L = [n]$, the set of vertices on the right is $R = [m]$, and vertex i on the right is connected with vertices $G_{i,1}, \ldots, G_{i,d}$ on the left. The function $f = f_{P,G}$ is defined by

$$f(x)_i = P(x_{G_{i,1}}, \ldots, x_{G_{i,d}}) \text{ for each } i \in [m].$$

That is, we evaluate P over the neighbors of each right vertex.

As we represent the graph as $G \in [n]^{m \times d}$, we are implicitly imposing an ordering on the set of edges going out of each vertex on the right.

An algorithm $\mathcal{A}$ for inverting the function f is said to be successful given some $b \in \{0,1\}^m$ if it can return some $\mathcal{A}(b) \in \{0,1\}^n$ such that $f(\mathcal{A}(b)) = b$.

Definition 2.2 *Fix a* d*-ary predicate* P*. We say that Goldreich's function* $f_{P,G}$ *with random graph* G *is* **secure** *against a class of algorithms* $\mathcal{A}$ *if with probability* $\geqslant 1-2^{-\Theta(n)}$ *over random uniform choice of a bipartite graph* G *with* $m = n$ *right nodes of degree* d*, for every algorithm* $\mathcal{A}$ *in that class of algorithms, only with probability* $\leqslant 2^{-\Theta(n)}$ *over random uniform choice of* $x \in \{0,1\}^n$ *is* $\mathcal{A}$ *given* $b = f_{P,G}(x)$ *successful in time* $\leqslant 2^{\Theta(n)}$*. The hidden constants in the asymptotic notation* $\Theta(n)$ *in this definition may depend on* d*.*

2.2 Expected Size of the Preimage

Definition 2.3 *We define the* **expected size of the preimage** *under* $f : \{0,1\}^n \to \{0,1\}^m$ *as*

$$\underset{x \sim \mathrm{Unif}(\{0,1\}^n)}{\boldsymbol{E}}[|f^{-1}(f(x))|].$$

When this expected size is M, we can think that the function f is M-to-one on average. We expect that for a good one-way function, M is not too large.

Theorem 2.4 *Whenever the predicate* P *passes a certain test, Goldreich's* $f_{P,G}$ *satisfies*

$$\underset{G \sim \mathrm{Unif}([n]^{n \times d})}{\boldsymbol{E}} \; \underset{x \sim \mathrm{Unif}(\{0,1\}^n)}{\boldsymbol{E}}[|f_{P,G}^{-1}(f_{P,G}(x))|] \leqslant n^{O(1)} 2^{2^{-\Omega(d)} n}.$$

This test

(A) *is satisfied by a uniformly random predicate with probability* $1 - o_d(1)$*;*

(B) *is satisfied by predicates of the form* $P = x_1 \oplus \ldots \oplus x_{d-h} \oplus Q(x_{d-h+1}, \ldots, x_d)$ *when* $d - h$ *is any function of* d *which is* $\Omega(d)$*;*

(C) *can be performed in time* $2^{O(d)}$*.*

2.3 Backtracking Algorithms

Definition 2.5 *At each step of a* **backtracking algorithm** $\mathcal{A}$ *for inverting Goldreich's function, i.e. finding* x *such that* $f(x) = b$*, a subset of the variables* $x_1, \ldots, x_n$ *have been assigned binary values and the rest of the variables are free. At the beginning of the backtracking algorithm, all variables are free. The backtracking algorithm* $\mathcal{A}$ *stops searching a path further if for some* i*, all the variables* $x_{G_{i,1}}, \ldots, x_{G_{i,d}}$ *have values assigned to them and* $P(x_{G_{i,1}}, \ldots, x_{G_{i,d}}) \neq b_i$*. Otherwise,* $\mathcal{A}$ *chooses an assignment* $x_j \leftarrow a$ *where* x_j *is a free variable and* $a \in \{0,1\}$ *is a value for* x_j*. The algorithm first assigns the value* a *to variable* x_j *and recursively calls the algorithm. If the algorithm* $\mathcal{A}$ *does not find* x *such that* $f(x) = b$ *in this recursion, it recurses again this time assigning* $1 - a$ *to* x_j*. The algorithm stops when there are no free variables and* $f(x) = b$*.*

We consider "drunk" and "myopic" backtracking algorithms.

2.4 Drunk Backtracking Algorithm

Definition 2.6 *A* **drunk** *backtracking algorithm* $\mathcal{A}$ *for inverting* $f_{P,G}$ *is one in which at each step, when choosing the pair* (j, a)*, the index* j *of variable* x_j *is allowed to be chosen by algorithm* $\mathcal{A}$ *from among any of the free variables* x_j*, but the value* $a \in \{0,1\}$ *has to be chosen each time using an independent unbiased binary coin flip.*

Theorem 2.7 *Goldreich's function* $f_{P,G}$ *with random graph* G *is secure against the class of drunk backtracking algorithms whenever the predicate* P *passes a test that satisfies the properties (A) and (C) of Theorem 2.4 together with a new property (B'). Property (B') says that for every constant* $0 \leqslant c < \frac{1}{2}$ *there exists a lower bound* D *such that whenever* $d \geqslant D$*, the test is satisfied by predicates of the form* $P = x_1 \oplus \ldots \oplus x_{d-h} \oplus Q(x_{d-h+1}, \ldots, x_d)$ *for* $h \leqslant cd$.

2.5 Myopic Backtracking Algorithm

Definition 2.8 *A* **myopic** *backtracking algorithm* $\mathcal{A}$ *for inverting* $f_{P,G}$ *is one which always has a limited view of the given* $b \in \{0,1\}^m$ *for which it wants to find an* $x \in f^{-1}_{P,G}(b)$*. Basically, at each step, the algorithm knows a subset of the bits of* b*. On the other hand, the algorithm has full knowledge of* P *and of* G*. In the beginning,* $\mathcal{A}$ *knows nothing about* b*. At each step, the algorithm is allowed to*

– *either query a bit of* b*,*
– *or choose a pair* (j, a)*; then assign the value* a *to* x_j *and recurse; if no* $x \in f^{-1}(b)$ *is found during this recursion, then assign* $1 - a$ *to* x_j *and recurse.*

When the algorithm recurses, the knowledge of b *is preserved; but the knowledge of* b *is not passed from one recursion tree to a sibling recursion tree. In other words, when we return back from a recursion, we leave behind all the knowledge we gained.*

Definition 2.9 *A myopic backtracking algorithm is called* (s,t)*-myopic if it never tries to read more than* t *bits of* b *before it has assigned binary values to at least* s *variables.*

Theorem 2.10 *There is* $t = \Theta(n)$ *such that for any* s *such that* $s/n = 2^{-o(d)}$*, Goldreich's function* $f_{P,G}$ *with random graph* G *is secure against the class of* (s,t)*-myopic backtracking algorithms whenever the predicate* P *passes a test that satisfies all the properties (A), (B'), and (C) in Theorem 2.7. The hidden constant in the asymptotic* $\Theta(n)$ *notation depends on* d.

2.6 Backtracking Algorithms with DPLL Elimination Rules

Definition 2.11 *Assume we are in the middle of a backtracking algorithm trying to find an element* x *in the preimage of* b *under* $f_{P,G}$*: A subset of the variables* $x_1, \ldots, x_n$ *have been assigned binary values and the rest of the variables are free. Let* x_j *be a free variable and let* $a \in \{0,1\}$*. We call* $x_j \leftarrow a$ *a* **DPLL assignment** *if, given all the already-assigned variables, one of the following two situations happens:*

– $x_j = a$ *is implied from the equation* $b_i = P(x_{G_{i,1}}, \ldots, x_{G_{i,d}})$ *for some* i*; in this case we say that a* **unit clause** *exists.*
– *for all* $i = 1, \ldots, m$*, switching the value of* x_j *from* $1 - a$ *to* a *can never change the equations* $b_i = P(x_{G_{i,1}}, \ldots, x_{G_{i,d}})$ *to become false; in this case we say that a* **pure literal** *exists.*

(The terms "unit clause" and "pure literal" come from the context of CNF formulas: We have adapted the terminology in the context of Goldreich's function.) When $x_j \leftarrow a$ is a DPLL assignment, conditioned on the already assigned values for the variables x_1, …, x_n that are not free, if there is no x in the preimage with $x_j = a$, neither is there an x in the preimage with $x_j = 1 - a$.

Definition 2.12 *A DPLL backtracking algorithm is similar to an ordinary backtracking algorithm except that at each step, for any existing DPLL assignment $x_j \leftarrow a$, we have the option of eliminating x_j from the list of free variables by assigning a to x_j and recursing (and when this recursion finishes, we do not try assigning $1-a$ to x_j.)*

DPLL drunk backtracking algorithms are simply drunk backtracking algorithms that are allowed to eliminate as above free variables x_j in DPLL assignments $x_j \leftarrow a$. Defining DPLL myopic backtracking algorithms involves a certain subtlety: Having access to the set of DPLL assignment $x_j \leftarrow a$ might reveal extra information about b. However, we allow a DPLL myopic backtracking algorithm to know the set of DPLL assignments. Yet, we allow this knowledge to be only passed further down in recursions; we do not allow such knowledge to be passed from a recursion tree to a sibling recursion tree. Besides knowing the set of DPLL assignments, an (s,t)-myopic DPLL backtracking algorithm for finding $x \in f^{-1}(b)$ may not read more than t bits of b before making at least s *non-DPLL* assignments to x.

Theorem 2.13 *Theorem 2.7 and Theorem 2.10 respectively hold for DPLL drunk and DPLL myopic backtracking algorithms as well.*

This theorem, as stated in this section, gives super-polynomial lower bounds on the running time of drunk and myopic DPLL backtracking algorithms for inverting Goldreich's function with non-negligible probability, when the predicate passes the tests mentioned in Theorems 2.7 and 2.10, if d is large enough but still constant compared to n. However, if one looks at the proof of the theorem, the same result holds when $d = o(\log n)$ or when $d \leqslant c \log n$ for suitably small positive constant c.

3 Expected Size of the Preimage

Inverting a candidate one-way function $f : \{0,1\}^n \to \{0,1\}^m$ is the problem of finding, given $b \in \{0,1\}^m$, an element in the preimage $f^{-1}(b)$, where $b = f(x)$ for $x \in \{0,1\}^n$ chosen uniformly at random. The problem of finding x itself given b is a harder problem, but as the following lemma shows, these two problems are related via the expected size of the preimages under f.

Lemma 3.1 *Let $f : \{0,1\}^n \to \{0,1\}^m$ be a function with expected preimage size $M = \boldsymbol{E}_{x \in \{0,1\}^n} |f^{-1}(f(x))|$. Let $\mathcal{A}$ be an algorithm that, given $b \in \{0,1\}^m$, returns $\mathcal{A}(b) \in \{0,1\}^n$. Choose x uniformly at random from $\{0,1\}^n$ and let $b = f(x)$. Consider the events $E = \{x : \mathcal{A}(b) = x\}$, and $F = \{x : \mathcal{A}(b) \in f^{-1}(b)\}$. We have*

$$\Pr[F] \leqslant 2\sqrt{M \Pr[E]}.$$

Proof. Consider the event $H = \{x : |f^{-1}(b)| \leqslant M'\}$, where we will specify $M' > 0$ shortly. We have $\Pr[F] \leqslant \Pr[F, H] + \Pr[H^c]$. To upper-bound $\Pr[H^c]$, we use Markov's inequality and get $\Pr[H^c] \leqslant M/M'$. We also have

$$\Pr[E] \geqslant \Pr[E|F,H] \cdot \Pr[F,H] \geqslant \frac{1}{M'} \cdot \Pr[F,H],$$

so we get $\Pr[F,H] \leqslant M' \Pr[E]$. We complete the proof by taking $M' = \sqrt{M/\Pr[E]}$, so that

$$\Pr[F] \leqslant \Pr[F,H] + \Pr[H^c] \leqslant M' \Pr[E] + M/M' = 2\sqrt{M \Pr[E]}.$$

□

The rest of this section is devoted to proving the upper bound of Theorem 2.4 on the expected size of the preimage for Goldreich's function. This theorem was shown for $h = 2$ and $Q(x, y) = x \wedge y$ in the conference version of this paper [8]. Itsykson [14] pointed out that the same proof works for general predicates Q for $h+1 < d/4$. The proof for random predicates is original to this work. We also mention that Panjwani [19] has previously done some experimental analysis of the size of preimages in Goldreich's function, composed with itself many times.

Theorem 2.4 concerns the expected size of preimages $|f_{P,G}^{-1}(f_{P,G}(x))|$ of Goldreich's function $f_{P,G}$. We assume the predicate P satisfies a test which we will define later, and choose the graph $G \in [n]^{n\times d}$ and the input $x \in \{0, 1\}^n$ uniformly at random.

We begin by observing that the expected size of the preimage M is equal to 2^{-n} times the number of colliding pairs of inputs:

$$\begin{aligned} M = & \mathop{\mathbf{E}}_{G\sim\mathrm{Unif}([n]^{n\times d})} \mathop{\mathbf{E}}_{x\sim\mathrm{Unif}(\{0,1\}^n)} [|f_{P,G}^{-1}(f_{P,G}(x))|] \\ =& 2^{-n} \mathop{\mathbf{E}}_{G\sim\mathrm{Unif}([n]^{n\times d})} |\{x, y \in \{0, 1\}^n : f_{P,G}(x) = f_{P,G}(y)\}| \end{aligned}$$

This allows us to express the expected preimage size in terms of collision probabilities:

$$\begin{aligned} M =& 2^{-n} \sum_{x,y\in\{0,1\}^n} \Pr_{G\sim\mathrm{Unif}([n]^{n\times d})} [f_{P,G}(x) = f_{P,G}(y)\}] \\ =& 2^{-n} \sum_{x,y\in\{0,1\}^n} \Pr_{G\sim\mathrm{Unif}([n]^{n\times d})} [\forall i \in [n],\ f_{P,G}(x)_i = f_{P,G}(y)_i] \end{aligned}$$

Recall that the i-th output bit of Goldreich's function $f_{P,G}$ depends on the input bits $x_{G_{i,1}}, \ldots, x_{G_{i,d}}$. Since the input indices $G_{i,1}, \ldots, G_{i,d}$ are chosen independently at random for each i, for a fixed x and y, the events $f_{P,G}(x)_i = f_{P,G}(y)_i$ for different values of i are independent:

$$M = 2^{-n} \sum_{x,y\in\{0,1\}^n} \prod_{i=1}^{n} \Pr_{G\sim\mathrm{Unif}([n]^{n\times d})} [f_{P,G}(x)_i = f_{P,G}(y)_i]$$

We introduce the notation $\mathrm{PE}_P(x, y) = \Pr_{G\sim\mathrm{Unif}([n]^{n\times d})}[f_{P,G}(x)_i = f_{P,G}(y)_i]$ (for **P**robability of **E**quality) noticing that the value is the same for every index i:

$$M = \sum_{x,y\in\{0,1\}^n} (2^{-1}\,\mathrm{PE}_P(x, y))^n \tag{1}$$

Now, suppose there are n_{ab} indices j such that $x_j = a$ and $y_j = b$; $n_{00} + n_{01} + n_{10} + n_{11} = n$. Then $\mathrm{PE}_P(x, y)$ depends only on the predicate P and the four numbers n_{ab}, since the equality $f_{P,G}(x)_i = f_{P,G}(y)_i$ depends only on the pairs of bits $(x_{G_{i,1}}, y_{G_{i,1}}), \ldots, (x_{G_{i,d}}, y_{G_{i,d}})$. This motivates the following definition.

Definition 3.2

– *For a pair* $(x, y) \in (\{0,1\}^n)^2$*, define* NA *(for Number of Appearances) by* $\mathrm{NA}(x, y) = (n_{00}, n_{01}, n_{10}, n_{11})$*, where* n_{ab} *is the number of indices* i *such that* $x_i = a$ *and* $y_i = b$*. If we let* $\Delta^2(n) = \{\beta : \{0,1\}^2 \to \mathbb{N} \mid \sum \beta = n\}$ *be the set of ways of putting* n *balls into four bins, then* $\mathrm{NA}(x, y) \in \Delta^2(n)$*. Furthermore, if we let* $\Delta^2 = \{p : \{0,1\}^2 \to \mathbb{R}^{\geqslant 0} \mid \sum p = 1\}$ *be the set of probability distributions over* $\{0,1\}^2$*, then* $\mathrm{NA}(x, y)/n \in \Delta^2$*. For example,* $\mathrm{NA}(0110, 1000)/4 = (1/4, 1/4, 1/2, 0)$.

– *For* $\alpha \in \Delta^2$*, we define* α^d *to be the distribution over* $(\{0,1\}^d)^2$ *such that* $(x, y) \sim \alpha^d$ *means each pair* (x_i, y_i) *is distributed according to* α *independently. For example, if* α *is the uniform distribution,* α^d *is also the uniform distribution, and if* α *assigns a probability of* 1 *to the string* 01*, then* α^d *assigns a probability of* 1 *to the pair* $(0 \cdots 0, 1 \cdots 1)$*. Finally,* $\mathcal{H}(\alpha)$ *denotes the base-2 entropy of the distribution:* $\mathcal{H}(\alpha) = -\sum_{i,j \in \{0,1\}} \alpha_{i,j} \lg \alpha_{i,j}$.

The value $\mathrm{PE}_P(x, y)$ depends only on P and the normalized number of occurrences $\mathrm{NA}(x, y)/n$.

Definition 3.3 *For a predicate* $P : \{0,1\}^d \to \{0,1\}$ *and a probability distribution* $\alpha \in \Delta^2$*, the probability of equality of* P *over* α *is*

$$\mathrm{PE}_P(\alpha) = \Pr_{(x,y)\sim\alpha^d}[P(x) = P(y)].$$

This allows us to rewrite (1) as

$$M = \sum_{x,y \in \{0,1\}^n} (2^{-1}\,\mathrm{PE}_P(\mathrm{NA}(x, y)/n))^n$$

In the above expression, $\mathrm{NA}(x, y)$ takes on values in $\Delta^2(n)$ depending on the strings x and y. For any particular $\beta \in \Delta^2(n)$, the number of pairs $x, y \in \{0,1\}^n$ satisfying $\mathrm{NA}(x, y) = \beta$ is equal to $\binom{n}{\beta_{00},\beta_{01},\beta_{10},\beta_{11}}$. So we have:

$$M = \sum_{\beta \in \Delta^2(n)} \binom{n}{\beta_{00}, \beta_{01}, \beta_{10}, \beta_{11}} (2^{-1}\,\mathrm{PE}_P(\beta/n))^n$$

(using Stirling's approximation)

$$\begin{aligned} &\leqslant \sum_{\beta \in \Delta^2(n)} O((2^{\mathcal{H}(\beta/n)} \cdot 2^{-1}\,\mathrm{PE}_P(\beta/n))^n) \\ &\leqslant |\Delta^2(n)| \max_{\alpha \in \Delta^2} O((2^{\mathcal{H}(\alpha)-1}\,\mathrm{PE}_P(\alpha))^n) \\ &= O(n^3) \max_{\alpha \in \Delta^2} (2^{\mathcal{H}(\alpha)-1}\,\mathrm{PE}_P(\alpha))^n. \end{aligned}$$

Therefore, in order to prove the lower bound of Theorem 2.4 that $M \leqslant n^{O(1)} 2^{2^{-\Omega(d)} n}$, we have only to show:

$$\forall \alpha \in \Delta^2,\ \mathcal{H}(\alpha) + \lg \mathrm{PE}_P(\alpha) \leqslant 1 + 2^{-\Omega(d)}. \tag{2}$$

In sections 3.1 and 3.2, we show that most random predicates and predicates of the form $P_{h,Q} = x_1 \oplus \ldots \oplus x_{d-h} \oplus Q(x_{d-h+1}, \ldots, x_d)$ satisfy (2). It is possible in time $2^{O(d)}$ to test whether a predicate has the form $P_{h,Q}$, or whether it has the properties we demand of random predicates in Section 3.2, so our proof is complete.

3.1 Proof of (2) for $P = x_1 \oplus \ldots \oplus x_{d-h} \oplus Q(x_{d-h+1}, \ldots, x_d)$

For predicates of the above form, we have

$$\begin{aligned}
\mathrm{PE}_P(\alpha) &= \frac{1 + \mathbf{E}[(-1)^{P(x)+P(y)}]}{2} \\
&= \frac{1 + \left(\prod_{i=1}^{d-h} \mathbf{E}[(-1)^{x_i+y_i}]\right) \mathbf{E}[(-1)^{Q(x_{d-h+1},\ldots,x_d)+Q(y_{d-h+1},\ldots,y_d)}]}{2} \\
&\leqslant \frac{1 + \left|\prod_{i=1}^{d-h} \mathbf{E}[(-1)^{x_i+y_i}]\right|}{2} \\
&= \frac{1 + |\alpha_{00} + \alpha_{11} - \alpha_{10} - \alpha_{01}|^{d-h}}{2}.
\end{aligned}$$

Let $p = \min\{\alpha_{00} + \alpha_{11}, \alpha_{01} + \alpha_{10}\}$. Then:

$$\begin{aligned}
\mathrm{PE}_P(\alpha) &\leqslant \frac{1 + (1-2p)^{d-h}}{2} \\
&\leqslant \frac{1 + (1-p)^{d-h}}{2}.
\end{aligned}$$

Given p, the maximum value of $\mathcal{H}(\alpha)$ is achieved when $\alpha_{00} = \alpha_{11}$ and $\alpha_{01} = \alpha_{10}$; thus $\mathcal{H}(\alpha) \leqslant 1 + \mathcal{H}(p)$. Therefore, to prove $\mathcal{H}(\alpha) + \lg \mathrm{PE}_P(\alpha) \leqslant 1 + 2^{-\Omega(d)}$, it suffices to show $\mathcal{H}(p) + \lg(1 + (1-p)^{d-h}) \leqslant 1 + 2^{-\Omega(d)}$. As long as $d - h = \Omega(d)$, the following lemma completes the proof with $\tau = (d-h)/d$.

Lemma 3.4 $\forall \tau \in (0, 1]$, $\exists \epsilon > 0$, $\forall p \in [0, 1]$ $\forall d \geqslant 1$, $\mathcal{H}(p) + \lg(1 + (1-p)^{\tau d}) \leqslant 1 + 2^{-\epsilon d}$.

The proof of this lemma can be found in Section 3.3.

3.2 Proof of (2) for a Random Predicate P

If Δ^2 were a small finite set, then we could prove (2) as follows. First, show that for any particular $\alpha \in \Delta^2$, a random predicate with probability $1 - \varepsilon$ satisfies $\mathcal{H}(\alpha) + \lg \mathrm{PE}_P(\alpha) \leqslant 1 + 2^{-\Omega(d)}$. Then with probability $1 - \varepsilon|\Delta^2|$ this is true for all $\alpha \in \Delta^2$ simultaneously, which is exactly (2). Since Δ^2 is an infinite set, this doesn't work, but our proof is similar in spirit: We will devise a small finite set of properties, each of which a random predicate P satisfies with high probability, then show that (2) follows from these properties.

In order to describe this set of properties, we begin with a new way to measure collisions under a predicate P. We have already seen $\mathrm{PE}_P(\alpha)$ where $\alpha \in \Delta^2$.

Definition 3.5 *For $\beta \in \Delta^2(d)$, the probability of equality of P over β is*

$$\mathrm{PE}_P(\beta) = \Pr_{(x,y) \sim \mathrm{NA}^{-1}(\beta)} [P(x) = P(y)],$$

where $\mathrm{NA}^{-1}(\beta)$ denotes the uniform distribution over the set of pairs $(x, y) \in (\{0,1\}^d)^2$ satisfying $\mathrm{NA}(x, y) = \beta$.

The two definitions of PE_P are related by $\mathrm{PE}_P(\alpha) = \mathbf{E}_{\beta \sim \mathrm{Mult}(\alpha,d)}[\mathrm{PE}_P(\beta)]$, where $\mathrm{Mult}(\alpha, d)$ denotes the multinomial distribution which draws d samples from $\{0,1\}^2$ according to the distribution α.

Definition 3.6 (One-Bit Entropy, α_{a*}, α_{*a}, $\mathcal{H}^*(\alpha)$) *Let $\alpha \in \Delta^2$.*

- *For $a \in \{0,1\}$, $\alpha_{a*} \overset{\text{def}}{=} \alpha(a,0) + \alpha(a,1)$ and $\alpha_{*a} \overset{\text{def}}{=} \alpha(0,a) + \alpha(1,a)$.*
- *We define the* one-bit entropy of α *to be* $\mathcal{H}^*(\alpha) \overset{\text{def}}{=} \max\{\mathcal{H}(\alpha_{0*}, \alpha_{1*}), \mathcal{H}(\alpha_{*0}, \alpha_{*1})\}$.

We now define a small set of properties from which (2) will follow.

Definition 3.7 (Collision-Averse Predicate) *For $\delta > 0$ and $\beta \in \Delta^2(d)$, we say P is* (δ, β)-collision averse *if*

$$\mathrm{PE}_P(\beta) \leqslant \tfrac{1}{2} + 2^{-d(\mathcal{H}^*(\beta/d)-\delta)/2}.$$

*Let $E(d)$ (for **E**qual) be the set of $\beta \in \Delta^2(d)$ such that $\beta(0,1) = \beta(1,0) = 0$. Notice that if $(x,y) \in \mathrm{NA}^{-1}(\beta)$, then $x = y$ iff $\beta \in E(d)$.*

If P is (δ,β)-collision averse for every $\beta \in \Delta^2(d) \setminus E(d)$, we say P is δ-collision averse.

Lemma 3.8 *Fix any $\delta > 0$ and $\beta \in \Delta^2(d) \setminus E(d)$. Choose $P : \{0,1\}^d \to \{0,1\}$ uniformly at random. Then*

$$\Pr[P \textit{ is not } (\delta,\beta)\textit{-collision averse}] \leqslant \exp(-\tfrac{1}{2}2^{\delta d}/\mathit{poly}(d)).$$

Taking a union bound,

$$\Pr[P \textit{ is not } \delta\textit{-collision averse}] \leqslant \exp(-2^{\delta d - O(\log d)}).$$

Proof (Lemma 3.8). Without loss of generality, assume $\mathcal{H}^*(\beta/d) = \mathcal{H}(\beta_{0*}/d, \beta_{1*}/d)$. Let $S \subseteq \{0,1\}^d$ be the support of the marginal distribution on x when $(x,y) \sim \mathrm{NA}^{-1}(\beta)$: that is, S is the set of strings with β_{0*} zeroes and β_{1*} ones.

Pick any $x \in S$. If $P, P' : \{0,1\}^d \to \{0,1\}$ are predicates which differ only at x, then $\mathrm{PE}_{P'}(\beta) - \mathrm{PE}_P(\beta) \leqslant c_x$, where

$$c_x = \Pr_{(x',y') \sim \mathrm{NA}^{-1}(\beta)}[x' = x \vee y' = x] \leqslant \frac{2}{|S|} = \frac{2}{\binom{d}{\beta_{0*}}}.$$

Fix P arbitrarily on all $x \notin S$, but choose the value of P independently at random for all $x \in S$: then $\mathbf{E}[\mathrm{PE}_P(\beta)] = \frac{1}{2}$, since $\beta \notin E(d)$. By McDiarmid's inequality, for any ϵ,

$$\begin{aligned}\Pr[\mathrm{PE}_P(\beta) > \tfrac{1}{2} + \epsilon] &\leqslant \exp\left(-\frac{2\epsilon^2}{\sum_{x \in S} c_x^2}\right) \\ &\leqslant \exp\left(-\tfrac{1}{2}\epsilon^2 \binom{d}{\beta_{0*}}\right) \\ &\leqslant \exp\left(-\tfrac{1}{2}\epsilon^2 2^{d\mathcal{H}(\beta_{0*}/d, \beta_{1*}/d)}/\mathrm{poly}(d)\right).\end{aligned}$$

To complete the proof, take $\epsilon = 2^{-d(\mathcal{H}(\beta_{0*}/d, \beta_{1*}/d)-\delta)/2}$. □

Lemma 3.9 *There exists $\delta > 0$ such that Equation (2) holds for every δ-collision averse predicate. That is, there exist $\delta, \eta > 0$ and $D \in \mathbb{N}$ such that whenever $d > D$ and a d-ary predicate P is δ-collision averse,*

$$\forall \alpha \in \Delta^2,\ \mathcal{H}(\alpha) + \lg \mathrm{PE}_P(\alpha) \leqslant 1 + 2^{-\eta d}.$$

Proof. Let $\alpha \in \Delta^2$. By the concavity of the logarithm function,

$$\lg \mathrm{PE}_P(\alpha) \leqslant \lg \xi + \frac{1}{\xi \ln 2}(\mathrm{PE}_P(\alpha) - \xi)$$

where ξ will be determined later. Since $\mathrm{PE}_P(\alpha) = \mathbf{E}_{\beta\sim\mathrm{Mult}(\alpha,d)}[\mathrm{PE}_P(\beta)]$, we have

$$\lg \mathrm{PE}_P(\alpha) \leqslant \lg \xi + \frac{1}{\xi \ln 2}\left(\mathop{\mathbf{E}}_{\beta\sim\mathrm{Mult}(\alpha,d)} \mathrm{PE}_P(\beta) - \xi\right). \tag{3}$$

We assume P is δ-collision averse, where δ will be determined later. That means that whenever $\beta(0,1) + \beta(1,0) > 0$, it is the case that $\mathrm{PE}_P(\beta) \leqslant \frac{1}{2} + 2^{-d(\mathcal{H}^*(\beta/d)-\delta)/2}$. So, if we let $E(d) = \{\beta : \beta(0,1) = \beta(1,0) = 0\}$ and $\gamma = \Pr_{\beta\sim\mathrm{Mult}(\alpha,d)}[\beta \in E(d)]$, we have:

$$\begin{aligned}\mathop{\mathbf{E}}_{\beta\sim\mathrm{Mult}(\alpha,d)} \mathrm{PE}_P(\beta) \leqslant& \gamma + (1-\gamma)\cdot \mathop{\mathbf{E}}_{\beta\sim\mathrm{Mult}(\alpha,d)}[\min\{1, \tfrac{1}{2} + 2^{-d(\mathcal{H}^*(\beta/d)-\delta)/2}\} | \beta \notin E(d)]\\ =& \tfrac{1+\gamma}{2} + (1-\gamma)\cdot \mathop{\mathbf{E}}_{\beta\sim\mathrm{Mult}(\alpha,d)}[\min\{\tfrac{1}{2}, 2^{-d(\mathcal{H}^*(\beta/d)-\delta)/2}\} | \beta \notin E(d)]\\ \leqslant& \tfrac{1+\gamma}{2} + \mathop{\mathbf{E}}_{\beta\sim\mathrm{Mult}(\alpha,d)}[\min\{\tfrac{1}{2}, 2^{-d(\mathcal{H}^*(\beta/d)-\delta)/2}\}].\end{aligned}$$

Whenever a random variable X is bounded above by 1, Markov's inequality gives $\mathbf{E}[X] \leqslant a + \Pr[X > a]$ for all $a \geqslant 0$. Taking $a = 2^{-\delta d/2}$ where the value of δ still has to be determined,

$$\mathop{\mathbf{E}}_{\beta\sim\mathrm{Mult}(\alpha,d)} \mathrm{PE}_P(\beta) \leqslant \tfrac{1+\gamma}{2} + 2^{-\delta d/2} + \mathop{\Pr}_{\beta\sim\mathrm{Mult}(\alpha,d)}[\mathcal{H}^*(\beta/d) < 2\delta].$$

Now, substitute back in to (3) taking $\xi = \frac{1+\gamma}{2}$:

$$\begin{aligned}\lg \mathrm{PE}_P(\alpha) \leqslant& \lg \tfrac{1+\gamma}{2} + \tfrac{2}{(1+\gamma)\ln 2}\left(2^{-\delta d/2} + \mathop{\Pr}_{\beta\sim\mathrm{Mult}(\alpha,d)}[\mathcal{H}^*(\beta/d) < 2\delta]\right)\\ \leqslant& \lg \tfrac{1+\gamma}{2} + 3\left(2^{-\delta d/2} + \mathop{\Pr}_{\beta\sim\mathrm{Mult}(\alpha,d)}[\mathcal{H}^*(\beta/d) < 2\delta]\right).\end{aligned}$$

Since $\mathcal{H}^*(\beta/d) \geqslant \mathcal{H}(\beta_{0*}/d)$, we have:

$$\lg \mathrm{PE}_P(\alpha) \leqslant \lg \tfrac{1+\gamma}{2} + 3\left(2^{-\delta d/2} + \mathop{\Pr}_{\beta\sim\mathrm{Mult}(\alpha,d)}[\mathcal{H}(\beta_{0*}/d) < 2\delta]\right).$$

Sampling $\beta \sim \mathrm{Mult}(\alpha, d)$ and looking at β_{0*} is the same as sampling $k \sim \mathrm{Binom}(\alpha_{0*}, d)$. So we can rewrite the above as:

$$\lg \mathrm{PE}_P(\alpha) \leqslant \lg \tfrac{1+\gamma}{2} + 3\left(2^{-\delta d/2} + \mathop{\Pr}_{k\sim\mathrm{Binom}(\alpha_{0*},d)}[\mathcal{H}(k/d) < 2\delta]\right). \tag{4}$$

Since $\lg \mathrm{PE}_P(\alpha) \leqslant 0$, the claim of the lemma follows when $\mathcal{H}(\alpha) \leqslant 1$; so henceforth we assume $\mathcal{H}(\alpha) \geqslant 1$, and hence $\mathcal{H}^*(\alpha) \geqslant 1/2$. We may further assume without loss of generality that $\mathcal{H}(\alpha_{0*}) \geqslant 1/2$. Let $p_0 \in (0, \frac{1}{2})$ be the unique number satisfying $\mathcal{H}(p_0) = \frac{1}{2}$; we know $\alpha_{0*} \in [p_0, 1-p_0]$. We now introduce a technical lemma which is proved in Section 3.3:

Lemma 3.10 *For every $p_0 \in (0, \frac{1}{2}]$, there exists $\delta > 0$ such that for all sufficiently large $d \in \mathbb{N}$,*

$$\forall p \in [p_0, 1-p_0], \quad \Pr_{k \sim \mathrm{Binom}(p,d)}[\mathcal{H}(k/d) < 2\delta] + 2^{-\delta d/2} \leqslant 2^{-\delta d/3}.$$

Apply this lemma to get $\delta > 0$, and substitute back into (4):

$$\lg \mathrm{PE}_P(\alpha) \leqslant \lg(1+\gamma) - 1 + 3 \cdot 2^{-\delta d/3}.$$

Now, the aim of this Lemma is to bound $\mathcal{H}(\alpha) + \lg \mathrm{PE}_P(\alpha)$. We have so far:

$$\begin{aligned}\mathcal{H}(\alpha) + \lg \mathrm{PE}_P(\alpha) \leqslant& \mathcal{H}(\alpha) + \lg(1+\gamma) - 1 + 3 \cdot 2^{-\delta d/3}\\ \leqslant& \mathcal{H}(\alpha_{00} + \alpha_{11}) + \lg(1+\gamma) + 3 \cdot 2^{-\delta d/3}.\end{aligned}$$

Now, $\gamma = \Pr[\beta(0,1) = \beta(1,0) = 0] = (\alpha_{00} + \alpha_{11})^d$. Apply Lemma 3.4, taking $\tau = 1$ and $p = 1 - (\alpha_{00} + \alpha_{11})$. Then:

$$\mathcal{H}(\alpha) + \lg \mathrm{PE}_P(\alpha) \leqslant 1 + 2^{-\epsilon d} + 3 \cdot 2^{-\delta d/3}.$$

Take $\eta < \min\{\epsilon, \delta/3\}$ and d sufficiently large that $2^{-\eta d} > 2^{-\epsilon d} + 3 \cdot 2^{-\delta d/3}$ to complete the proof. □

3.3 Proof of Lemmas 3.4 and 3.10

Proof of Lemma 3.4. Assume $\tau \in (0,1]$ is given. First we show that we can choose positive D and ϵ' such that

$$\forall p \in [0,1],\ \forall d > D,\ \mathcal{H}(p) + \lg(1 + (1-p)^{\tau d}) \leqslant 1 + 2^{-\epsilon' d}. \tag{5}$$

We prove this by considering four possible cases for the value of p, namely, $p \in (\epsilon_1, 1]$, $p \in (\epsilon_2/d, \epsilon_1]$, $p \in (2^{-\epsilon_3 d}, \epsilon_2/d]$, $p \in [0, 2^{-\epsilon_3 d}]$, where ϵ_1, ϵ_2, ϵ_3 are positive constants to be chosen. We will choose the numbers D and $\epsilon', \epsilon_1, \epsilon_2, \epsilon_3$ as we go along, but according to the following dependency graph:

$$\begin{tikzcd}
\epsilon_2 \arrow[r] & \epsilon_1 \arrow[d] \arrow[dr] & \\
& \epsilon' & D \\
& \epsilon_3 \arrow[u] \arrow[ur] &
\end{tikzcd}$$

– Case 1: $p > \epsilon_1$. Then

$$\mathcal{H}(p) + \lg(1 + (1-p)^{\tau d}) \leqslant 1 + (1-\epsilon_1)^{\tau d} \lg e \leqslant 1 + 2^{-\epsilon' d},$$

for $\epsilon_1 < 1$, $\epsilon' \leqslant -\frac{1}{2}\tau \lg(1-\epsilon_1)$, $d > D \geqslant -2 \lg \lg e / (\tau \lg(1-\epsilon_1))$.

For the remaining three cases, p is small. Using the Taylor expansion of lg around 2, we get

$$\lg(1 + (1-p)^{\tau d}) \leqslant 1 + \frac{(1-p)^{\tau d} - 1}{2 \ln 2} \leqslant 1 + \frac{e^{-\tau p d} - 1}{2 \ln 2}.$$

– Case 2: $p \in (\epsilon_2/d, \epsilon_1]$. Then

$$\mathcal{H}(p) + \lg(1 + (1-p)^{\tau d}) \leqslant \mathcal{H}(\epsilon_1) + 1 + \frac{e^{-\tau\epsilon_2} - 1}{2\ln 2} \leqslant 1,$$

if we choose ϵ_1 small enough that $\epsilon_1 \leqslant 1/2$ and $\mathcal{H}(\epsilon_1) \leqslant (1 - e^{-\tau\epsilon_2})/(2\ln 2)$.

For the remaining two cases we fix $\epsilon_2 = (2\tau)^{-1}$. Now, $\tau p d \leqslant \frac{1}{2}$, and we have the approximation

$$\mathcal{H}(p) + 1 + \frac{e^{-\tau p d} - 1}{2\ln 2} \leqslant (p\lg(1/p) + 2p) + 1 - \frac{\tau p d}{4\ln 2} = 1 + p\left(\lg(1/p) - \frac{\tau}{4\ln 2}d + 2\right).$$

– Case 3: $p \in (2^{-\epsilon_3 d}, \epsilon_2/d]$.
For $\epsilon_3 < \frac{\tau}{4\ln 2}$ and $d > D$ for sufficiently large D (depending on ϵ_3): $\lg(1/p) - \frac{\tau}{4\ln 2}d + 2 < 0$.

– Case 4: $p \leqslant 2^{-\epsilon_3 d}$.
For $\epsilon' \leqslant \frac{1}{2}\epsilon_3$ and $d > D$ for sufficiently large D (depending on ϵ_3): $p\lg(1/p) \leqslant \epsilon_3 d 2^{-\epsilon_3 d} \leqslant 2^{-\epsilon' d}$.

We have proved (5). It remains to prove the lemma for $d \in [1, D]$. Let $f(p, d) = \mathcal{H}(p) + \lg(1+(1-p)^{\tau d})$. Since f is a continuous function on the compact set $[0, 1] \times [1, D]$, it achieves a finite maximum $M = f(p_*, d_*)$ on this set. It is easy to see that $M \in (1, 2)$. Let $\epsilon = \min\{\epsilon', -D^{-1}\lg(M-1)\}$. Then for $d \in [1, D]$, $f(p, d) \leqslant M \leqslant 1+2^{-\epsilon d}$, and for $d \in (D, \infty)$, $f(p, d) \leqslant 1 + 2^{-\epsilon' d} \leqslant 1 + 2^{-\epsilon d}$. □

Proof of Lemma 3.10. Let D_{KL} denote the Kullback-Leibler divergence

$$D_{KL}(q\|p) = q\lg\frac{q}{p} + (1-q)\lg\frac{1-q}{1-p}.$$

Fix $\lambda > 0$ to be small enough that $\lambda < 1/2$ and for any $p \in [p_0, 1 - p_0]$,

$$D_{KL}(\lambda\|p) > \mathcal{H}(\lambda).$$

Now, choose any $p \in [p_0, 1 - p_0]$.

$$\begin{aligned}
\Pr_{k\sim\text{Binom}(p,d)}[\mathcal{H}(k/d) < \mathcal{H}(\lambda)] &= \Pr[k/d < \lambda] + \Pr[k/d > 1 - \lambda] \\
\textit{(Without loss of generality, assume } p \leqslant \tfrac{1}{2}.\textit{)} \qquad &\leqslant 2\Pr[k/d < \lambda] \\
\textit{(Apply Chernoff's bound.)} \qquad &\leqslant 2\exp(-D_{KL}(\lambda\|p)d) \\
&< 2\exp(-\mathcal{H}(\lambda)d).
\end{aligned}$$

Complete the proof by taking $\delta = \mathcal{H}(\lambda)/2$ and taking d to be sufficiently large that $2\exp(-\mathcal{H}(\lambda)d) < 2^{-\delta d/3} - 2^{-\delta d/2}$. □

4 Locally Consistent Partial Assignment

In this section, we assume we are given an instance of Goldreich's function $f = f_{P,G} : \{0, 1\}^n \to \{0, 1\}^m$ together with $b \in \{0, 1\}^m$. We want to analyze a backtracking algorithm $\mathcal{A}$ that tries to find an element in the preimage $f^{-1}(b)$. A crucial concept in this analysis will be that of a locally consistent partial assignment. In order to introduce this concept, we will introduce some preliminary concepts.

Definition 4.1 (Partial Assignment) *A* partial assignment *is a function* $\rho : [n] \to \{0, 1, *\}$. *Its set of* fixed variables *is* $\mathrm{Vars}(\rho) = \rho^{-1}(\{0,1\})$. *Its set of* free variables *is* $\rho^{-1}(\{*\})$. *Its* size *is defined to be* $|\rho| = |\mathrm{Vars}(\rho)|$. *Given* $f : \{0,1\}^n \to \{0,1\}^m$, *the* restriction of f by ρ, *denoted* $f|_\rho$, *is the function obtained by fixing the variables in* $\mathrm{Vars}(\rho)$ *and allowing the rest of the variables of* f *to vary.*

For a partial assignment ρ, *an index* $j \in [n]$, *and a value* $a \in \{0,1\}$, *we define the partial truth assignment* $\rho[x_j \leftarrow a]$ *by*

$$\rho[x_j \leftarrow a](i) = \begin{cases} a, & i = j; \\ \rho(i), & i \neq j. \end{cases}$$

Definition 4.2 (Boundary, Neighborhood, and Expansion) *Let* $G \in [n]^{m \times d}$ *be a bipartite graph as in Definition 2.1. Let* $I \subseteq R$ *be a subset of vertices on the right. Its* neighborhood $\Gamma(I) \subseteq L$ *is the set of all nodes adjacent to nodes in* I. *For* $i \in I$, *the* boundary of i in I, *denoted* $\partial_I i$, *is the set of nodes in* L *with one edge to* i *but no other edges to* I. *The* boundary *of* I, *denoted* ∂I, *is the set of all nodes* $j \in L$ *such that there is exactly one edge from* j *to* I. *Equivalently,* $\partial I = \bigcup_{i \in I} \partial_I i$.

G *is an* (r, c)-boundary expander *if for all* $I \subseteq R$ *such that* $|I| \leqslant r$, *we have* $|\partial I| \geqslant c|I|$.

Definition 4.3 (Closure) *Let* $G \in [n]^{m \times d}$ *be a bipartite graph as in Definition 2.1. Assume* G *is an* (r, c)*-boundary expander. Fix a subset of input nodes* $J \subseteq L$. *We say a subset of output nodes* $I \subseteq R$ *of size* $|I| \leqslant r/2$ *is a* closure *for* J *if the subgraph of* G *obtained by deleting nodes in* $J \cup \Gamma(I)$ *and nodes in* I *is an* $(r/2, c/2)$*-boundary expander.*

Note that a closure for a set of left-nodes is a set of right-nodes.

Definition 4.4 (Locally Consistent Partial Assignment) *Let* f *be Goldreich's function for graph* G *and predicate* P. *Let* $b \in \{0,1\}^m$ *and let* ρ *be a partial assignment. For a set of output nodes* $I \subseteq R$, *we say* ρ *is* consistent *with* I *if* ρ *can be extended to some* $x' \in \{0,1\}^n$ *such that* $f(x')_I = b_I$. *We say* ρ *is* locally consistent *if there exists a closure* I *for* $\mathrm{Vars}(\rho)$ *such that* ρ *is consistent with* I. *We say* ρ *is* globally consistent *if* ρ *is consistent with* R.

4.1 Backtracking Trees and Resolution Proofs

Definition 4.5 (Backtracking Tree) *The running of a backtracking algorithm* $\mathcal{A}$ *in finding a solution to the equation* $f_{P,G}(x) = b$ *can be modeled by a* backtracking tree. *Each node of the backtracking tree is labeled by a partial assignment* $\rho : [n] \to \{0, 1, *\}$, *where* $\rho_j = *$ *means variable* x_j *is free and* $\rho_j \neq *$ *means variable* x_j *is assigned the value* ρ_j. *We measure the* running time *of algorithm* $\mathcal{A}$ *in terms of the number of nodes in its backtracking tree.*

The root of the backtracking tree is labeled by the empty partial assignment $*^n$. *If at a step of the backtracking, the current partial assignment is* ρ *and algorithm* $\mathcal{A}$ *chooses the assignment* $x_j \leftarrow a$ *where* x_j *is a free variable and* $a \in \{0,1\}$, *then the left child of the current node is labeled by* $\rho[x_j \leftarrow a]$. *If the current node has a right child, the label of this right child is* $\rho[x_j \leftarrow 1 - a]$.

For example, here is part of a backtracking tree. One of the nodes is boxed. The label of that node is a partial assignment that assigns values 0, 1, 0 to variables x_5, x_7, and x_8 respectively.

$$\begin{tikzcd}
 & & & *^n \arrow[lld] & \\
 & {[x_5 = 0]} \arrow[ld, dashed] \arrow[rrd] & & & \\
 {} & & & {[x_5 = 0, x_7 = 1]} \arrow[lld] \arrow[rd, dashed] & \\
 & \boxed{[x_5 = 0, x_7 = 1, x_8 = 0]} \arrow[ld, dashed] \arrow[rd, dashed] & & & {} \\
 {} & & {} & &
\end{tikzcd}$$

In this subsection, we motivate the definition of locally consistent partial assignments by showing that a backtracking subtree with a root that is labeled by a locally consistent but globally inconsistent partial assignment has exponential size.

Definition 4.6 (Tree-like Resolution Proof) *If $C = a \vee D$ and $C' = \neg a \vee D'$ are two clauses each consisting of an OR of literals, C and C' together imply $D \vee D'$. In this case, $D \vee D'$ is called the* resolution *of C and C'. Let Φ be a CNF formula. A* resolution proof *for refuting Φ is a sequence of clauses $C_1, C_2, \dots, C_l$ where C_l is the empty clause, and for $i = 1, \dots, l$,*

- *either C_i is a clause appearing in Φ,*
- *or C_i is the resolution of two clauses that appear before C_i in the sequence.*

If each clause in the sequence (except the last clause C_l) is used exactly once to imply a later clause in the sequence through a resolution, then the resolution proof is called tree-like. *The length of the resolution proof is l. The width of the resolution proof is the width of the maximum-width clause among $C_1, \dots, C_l$.*

Lemma 4.7 (Refer to for example [16, Proposition 1]) *Let ρ be the label of the root of a subtree of a backtracking tree for finding a solution x to $f_{P,G}(x) = b$. One can express $f_{P,G}(x) = b$ by a CNF formula Φ by translating each constraint $P(x_{G_{i,1}}, \dots, x_{G_{i,d}}) = b_i$ into $\leqslant 2^d$ clauses of width d. Define $\Phi_{|\rho}$ to be the CNF formula which is the restriction of Φ by ρ. If no solution x is found in this subtree, then $\Phi_{|\rho}$ has a tree-like resolution proof of length no bigger than the number of nodes in the subtree.*

Proof. To each node labeled ρ' in this subtree, we can in the following recursive way associate a clause $C_{\rho'}$ over the free variables of ρ such that $C_{\rho'}$ is false under the partial assignment ρ':

- For a leaf node labeled ρ', one of the constraints $P(x_{G_{i,1}}, \dots, x_{G_{i,d}}) \neq b_i$ is not satisfiable under ρ'. Therefore at least one clause $C_{\rho'}$ exists in $\Phi_{|\rho}$ that is not satisfiable under ρ'.
- For a node labeled ρ' with left and right children $\rho^l = \rho'[x_j \leftarrow a]$ and $\rho^r = \rho'[x_j \leftarrow 1 - a]$, if either of the two clauses C_{ρ^l} or C_{ρ^r} does not include the variable x_j, then choose that clause as $C_{\rho'}$, otherwise let $C_{\rho'}$ be the resolution of C_{ρ^l} and C_{ρ^r}.

In this way, the labels of a postorder tree traversal of the subtree rooted at ρ can give a tree-like resolution proof for refuting $\Phi_{|\rho}$. □

Definition 4.8 (Robust Predicate) *Let* $0 \leqslant h < d$ *be an integer. The predicate* $P : \{0,1\}^d \to \{0,1\}$ *is* h-robust *if after any partial assignment of values to the input variables of* P *that lets* $> h$ *of the input variables of* P *be free, the predicate* P *can still take both of the two possible values of 0 and 1 as its value. For example, the predicate that sums all its inputs modulo* 2 *is* 0*-robust.*

Theorem 4.9 [6, Theorem 3.3] *The length of any tree-like resolution refutation of a CNF formula* Ψ *is at least* $2^{w - w_\Psi}$*, where* w *is the minimal width of a resolution refutation of* Ψ*, and* w_Ψ *is the maximal width of a clause in* Ψ*.*

Lemma 4.10 *Consider running a backtracking algorithm for solving* $f_{P,G}(x) = b$*. Assume that the algorithm reaches a locally consistent but globally inconsistent partial assignment* ρ*. Assume* P *is* h*-robust. Then in the backtracking tree, the subtree rooted at* ρ *has size at least* $2^{(c/2-h)r/4-d}$*.*

Proof. (Our proof follows the proof of [1, Lemma 8], which in turn uses the Ben-Sasson-Wigderson measure of [6].) Let $\Phi_{|\rho}$ be as in Lemma 4.7. By Theorem 4.9, it suffices to show that every resolution refutation of $\Phi_{|\rho}$ has width at least $w \overset{\text{def}}{=} (c/2-h)r/4$.

Since ρ is locally consistent, there exists a closure I for $\mathrm{Vars}(\rho)$ and ρ can be extended to some $x' \in \{0,1\}^n$ such that $f(x')_I = b_I$. Let $J = \mathrm{Vars}(\rho) \cup \Gamma(I)$. We will prove the stronger statement that every resolution refutation of $\Phi_{[x_J = x'_J]}$ has width at least w.

For a clause C on the variables $x_{L \setminus J}$ and a set $I' \subseteq R \setminus I$, we say I' *implies* C if for every x such that $(f(x)_{I'} = b_{I'} \wedge x_J = x'_J)$, the clause C is satisfied by x. We define the *measure* of C to be

$$\mu(C) = \min\{|I'| : \ I' \subseteq R \setminus I, \text{ and } I' \text{ implies } C\}.$$

Since ρ is globally inconsistent, this measure is well-defined. Assume $\mu(C) \leqslant r/2$, and let I' be a smallest subset of $R \setminus I$ which implies C. No vertex $i \in I'$ contains more than h neighbors in $\partial_{I'} i \setminus J$ that do not appear in C, since otherwise, by the h-robustness of the predicate P, then $I' \setminus \{i\}$ would also imply C. Since I is a closure for $\mathrm{Vars}(\rho)$, we know $|\partial I' \setminus J| \geqslant c|I'|/2$, so C consists of at least $(c/2-h)\mu(C)$ variables. Thus we have proved:

$$\text{For any clause } C,\ \mu(C) \text{ is either} \leqslant \frac{\mathrm{width}(C)}{c/2-h} \text{ or} > r/2. \qquad (\#)$$

We have:

1. $\mu(C) = 1$ for any clause C in the CNF formula $\Phi_{[x_J = x'_J]}$.
2. $\mu(C) > r/2$ for the empty clause $C = \text{False}$ (because of (#) and because the empty clause has positive measure $\mu(C)$.)
3. μ is subadditive: If C_2 is the resolution of C_0 and C_1, then $\mu(C_2) \leqslant \mu(C_0) + \mu(C_1)$, because whenever I'_0 implies C_0 and I'_1 implies C_1, it follows that $I'_0 \cup I'_1$ implies C_2.

Putting 1, 2 and 3 together, we find that every resolution refutation of $\Phi_{[x_J = x'_J]}$ contains a clause C whose measure is in the range $(r/4, r/2]$. By (#), the width of C is at least $w = (c/2-h)r/4$, which completes the proof. □

4.2 Clever Backtracking

For the analysis of drunk and myopic backtracking algorithms, we consider the left-most node in the backtracking tree among nodes labeled by locally consistent partial assignments. In order to argue about this particular partial assignment more easily, we modify the backtracking algorithm into a so-called *clever* backtracking algorithm such that this particular partial assignment appears on the left-most branch of the tree.

Definition 4.11 (Locally Forced Assignment) *Let f be Goldreich's function, let $b \in \{0,1\}^m$, let ρ be a partial assignment, and let $j \in [n] \setminus \mathrm{Vars}(\rho)$. We say an assignment $x_j \leftarrow a$ is* locally forced *if $\rho[x_j \leftarrow a]$ is locally consistent but $\rho[x_j \leftarrow 1-a]$ is not. Otherwise we say the assignment is locally* unforced*.*

Definition 4.12 (Clever Backtracking) *Let $\mathcal{A}$ be a backtracking algorithm for finding a solution x to $f_{P,G}(x) = b$. The algorithm $\mathcal{A}$ is* clever *if whenever the current partial assignment is ρ and the assignment $x_j \leftarrow 1-a$ is locally forced, the algorithm does not first try the assignment $x_j \leftarrow a$.*

Given a backtracking algorithm $\mathcal{A}$, we can give a clever version $\mathcal{C}(\mathcal{A})$ of $\mathcal{A}$ as follows: $\mathcal{C}(\mathcal{A})$ is similar to $\mathcal{A}$ except that at each step when the current partial assignment is ρ, if $\mathcal{A}$ chooses the assignment $x_j \leftarrow a$ to try first, the clever version $\mathcal{C}(\mathcal{A})$ checks whether $x_j \leftarrow 1-a$ is locally forced. If $x_j \leftarrow 1-a$ is locally forced, $\mathcal{C}(\mathcal{A})$ tries the assignment $x_j \leftarrow 1-a$ first instead of $x_j \leftarrow a$.

It is clear that the running time of the clever version of any backtracking algorithm $\mathcal{A}$ is as good as the algorithm $\mathcal{A}$ itself, since the clever algorithm only delays searching those subtrees that we are sure do not contain any solution in the preimage $f^{-1}(b)$.

Here is our main lemma about clever backtracking algorithms. We will defer its proof to the next section when we have developed the necessary tools.

Lemma 4.13 *Consider a clever backtracking algorithm $\mathcal{A}$. Assume the predicate P is h-robust for $h < c/2 - 1$. The algorithm will make at least $\lfloor cr/4 \rfloor$ locally unforced assignments on its leftmost branch. Furthermore, the partial assignment ρ obtained on this branch after $\lfloor cr/4 \rfloor$ locally unforced assignments is locally consistent.*

4.3 Some Properties of Closures and Consistency

Lemma 4.14 *If $I \subseteq R$ is a closure for $J \subseteq L$, then I is a closure also for $J \cup \Gamma(I)$.*

Proof. The statement follows directly from the definition of closure. □

Lemma 4.15 Analogous to [1, Lemma 6].
Let $J \subseteq L$ have size $|J| \leqslant cr/4$. Then there exists a closure C for J such that $|C| \leqslant 2c^{-1}|J|$.

Proof. Call $I \subseteq R$ *nonexpanding* if $|\partial I \setminus J| \leqslant c|I|/2$. For a nonexpanding I we have $|\partial I| \leqslant c|I|/2 + |J|$. If, furthermore, $|I| \leqslant r$, by the (r,c)-boundary-expansion of G we have $|\partial I| \geqslant c|I|$, so $|I| \leqslant 2c^{-1}|J| \leqslant r/2$. Therefore a nonexpanding $I \subseteq R$ has either $> r$ vertices or $\leqslant r/2$ vertices.

Let C be a largest nonexpanding set with $\leqslant r/2$ vertices. (C might be empty.) We claim that C is a closure for J. Indeed, let S be any subset of $R \setminus C$ with $\leqslant r/2$ vertices. It suffices to show that $|\partial S \setminus (J \cup \Gamma(C))| \geqslant c|S|/2$.

Suppose otherwise: then S is nonempty, and also $|\partial(C \cup S) \setminus J| \leqslant |\partial C \setminus J| + |\partial S \setminus (J \cup \Gamma(C))| < c|C|/2 + c|S|/2 = c|C \cup S|/2$. Then $C \cup S$ is a nonexpanding set with $\leqslant r$ vertices, and therefore $\leqslant r/2$ vertices. This contradicts our assumption that C was a largest nonexpanding set with $\leqslant r/2$ vertices.

Finally, we showed at the start of the proof that $|C| < 2c^{-1}|J|$. □

Lemma 4.16 Analogous to [1, Lemma 7].
A partial assignment ρ is consistent with all $I \subseteq R$ having size $|I| \leqslant r/2$ if ρ is locally consistent and P is h-robust for $h < c/2$. The same is true about ρ if ρ is consistent with a closure C for $\mathrm{Vars}(\rho) \setminus A$ where $A \subseteq \mathrm{Vars}(\rho)$ and P is h-robust for $h < c/2 - |A|$.

Proof. (We only prove the second statement of the Lemma: The statement about locally consistent ρ follows from the second statement by fixing $A = \emptyset$.)

Let I be a smallest set such that ρ is not consistent with I. Assume that, contrary to the statement of the lemma, $|I| \leqslant r/2$. We know $I' = I \setminus C$ is non-empty. Define $J = \mathrm{Vars}(\rho) \cup \Gamma(C)$. Since C is a closure for $\mathrm{Vars}(\rho) \setminus A$, we have $|\partial I' \setminus J| \geqslant c|I'|/2 - |A|$. In particular, there must be some $i \in I'$ with $|\partial_{I'} i \setminus J| \geqslant \lceil c/2 - |A| \rceil$.

Since I is a smallest set with which ρ is not consistent, ρ is consistent with $I \setminus \{i\}$. So extend ρ to a partial assignment x' which satisfies $(f(x'))_{I \setminus \{i\}} = b_{I \setminus \{i\}}$. Since P is a $\lceil c/2 - |A| - 1 \rceil$-robust predicate, we can modify input bits in the set $|\partial_{I'} i \setminus J|$ and leave all other input bits the same to produce an input x'' such that $(f(x''))_i = b_i$. Since x'' is equal to x' on every input bit in $\Gamma(I \setminus \{i\})$, we have $(f(x''))_I = b_I$. This contradicts the assumption that ρ is not consistent with I. □

We are now ready to prove Lemma 4.13.

Proof (of Lemma 4.13). Consider the running of clever algorithm $\mathcal{A}$ for s steps on its leftmost branch of its backtracking tree. Let $\rho^0, \ldots, \rho^s$, where $|\mathrm{Vars}(\rho^i)| = i$, be the sequence of the partial assignments appearing on the nodes of this leftmost branch. Let F_s be the set of indices $j \in \mathrm{Vars}(\rho^s)$ of variables x_j whose assignment by $\mathcal{A}$ during this branch was locally forced. Let $U_s = \mathrm{Vars}(\rho^s) \setminus F_s$.

We first prove by induction on s the following claim that

if C is a closure for U_s then $F_s \subseteq \Gamma(C)$, and thus by Lemma 4.14 C is a closure for $\mathrm{Vars}(\rho^s)$.

The base case $s = 0$ is obvious. Now assume the claim is true for $s - 1$, and we want to prove the claim for s. Since C is a closure for U_s, it is also a closure for the smaller set U_{s-1}, hence $F_{s-1} \subseteq \Gamma(C)$. If $F_{s-1} = F_s$, we are done with the proof of the claim. Otherwise, $\rho^s = \rho^{s-1}[x_j \leftarrow a]$, where ρ^s is locally consistent but $\rho^{s-1}[x_j \leftarrow 1 - a]$ is not locally consistent. ρ^s is consistent with a closure I for $\mathrm{Vars}(\rho^s)$. Thus, by Lemma 4.16, ρ^s is consistent with C. If $j \in \Gamma(C)$, we are done with the proof of the claim. Otherwise, $\rho' := \rho^{s-1}[x_j \leftarrow 1 - a]$ is also consistent with C. By an application of Lemma 4.16 with $A = \{j\}$, we know ρ' is consistent with I which is also a closure for $\mathrm{Vars}(\rho')$. This contradicts the fact that ρ' is not locally consistent, proving the claim.

In order to prove the Lemma itself, it suffices to prove the following two facts:

1. If $|U_s| \leqslant \lfloor cr/4 \rfloor$, there exists a closure C for $\mathrm{Vars}(\rho^s)$ (and therefore $s < n$).
2. If $|U_{s-1}| < \lfloor cr/4 \rfloor$ and ρ^{s-1} is locally consistent, then ρ^s is locally consistent.

To prove the first fact, notice that by Lemma 4.15 there exists a closure C for U_s. By the claim we proved earlier, C is also a closure for $\mathrm{Vars}(\rho^s)$.

To prove the second fact, let C be the closure for $\mathrm{Vars}(\rho^s)$ which is guaranteed to exist by the first fact. If ρ^s is not locally consistent, neither is $\rho^{s-1}[x_j \leftarrow 1-a]$ locally consistent because $\mathcal{A}$ is clever. Therefore, both $\rho^{s-1}[x_j \leftarrow a]$ and $\rho^{s-1}[x_j \leftarrow 1-a]$ are not consistent with C. But this exactly means ρ^{s-1} is not consistent with C; hence by Lemma 4.16, ρ^{s-1} is not locally consistent. This contradiction proves the second fact. $\square$

5 Drunk Backtracking Algorithms

The goal of this section is to prove Theorem 2.7.

5.1 Probability of Correct Guess

Notice that the clever version $\mathcal{C}(\mathcal{A})$ of a drunk algorithm $\mathcal{A}$ is also randomized whenever it makes a locally unforced assignment.

Lemma 5.1 (Main Drunk Lemma) *Let $\mathcal{A}$ be a drunk backtracking algorithm for inverting Goldreich's function. Assume that we are guaranteed that $\mathcal{C}(\mathcal{A})$ makes at least s' locally unforced assignments on the left-most branch of its backtracking tree as e.g. by Lemma 4.13. Choose $x' \in \{0,1\}^n$ and let $b = f_{P,G}(x')$. When b is given as input to $\mathcal{C}(\mathcal{A})$, let $\rho^{s'}$ be the resulting partial assignment on the left-most branch after s' locally unforced assignments. The probability, over the randomness of algorithm $\mathcal{C}(\mathcal{A})$, that $\rho^{s'}$ can be extended to x' is exactly $2^{-s'}$.*

Proof. We shall prove this by induction on s'. The statement for the base case $s' = 0$ is clear: $*^n$ can always be extended to x.

For $s' \geqslant 1$, let $\rho^{s'}_{\mathrm{bef}}$ be the partial assignment just before the s'-th non-forced assignment is made. Since $\rho^{s'}_{\mathrm{bef}}$ is obtained from $\rho^{s'-1}$ by adding some locally forced assignments, we know that $\rho^{s'}_{\mathrm{bef}}$ can be extended to x' if and only if $\rho^{s'-1}$ can be.

Furthermore, to get from $\rho^{s'}_{\mathrm{bef}}$ to $\rho^{s'}$, the drunk algorithm $\mathcal{A}$ makes a choice $x_j \leftarrow a$ where $\Pr[a = 0] = \Pr[a = 1] = 1/2$. Since this step is not locally forced, the clever version $\mathcal{C}(\mathcal{A})$ allows first trying the choice of a for x_j, which is not equal to x'_j with probability 1/2. Therefore,

$$\begin{aligned}\Pr[\rho^{s'} \text{ can be extended to } x'] =& \Pr[\rho^{s'}_{\mathrm{bef}} \text{ can be extended to } x'] \ \Pr[a = x_i | \rho^{s'}_{\mathrm{bef}} \text{ can be extended to } x'] \\ =& \Pr[\rho^{s'-1} \text{ can be extended to } x'] \cdot 1/2 \\ =& 2^{-s'+1} \cdot 1/2 = 2^{-s'}.\end{aligned}$$

$\square$

5.2 Choice of Predicate and Graph

For Lemmas 4.10 and 4.13, we need that the predicate P be h-robust for small h. The following lemma provides two classes of predicates that satisfy this.

Lemma 5.2 *The predicate $P = x_1 \oplus \ldots \oplus x_{d-h} \oplus Q(x_{d-h+1}, \ldots, x_d)$ is h-robust. A random predicate P on d variables is $\Theta(\log d)$-robust with probability $1 - o_d(1)$.*

Proof. Predicates of the form $P = x_1 \oplus \ldots \oplus x_{d-h} \oplus Q(x_{d-h+1}, \ldots, x_d)$ are h-robust, since any subset of $h+1$ variables includes at least one of the variables $x_1, \ldots, x_{d-h}$.

Now assume P is a random d-ary predicate. Let ρ be any partial assignment on d variables which fixes all but $h+1$ variables. Let E_ρ be the event that the predicate P becomes constant under the partial assignment ρ. We have $\Pr[E_\rho] = 2 \cdot 2^{-2^{h+1}} = 2^{1-2^{h+1}}$. P is h-robust if for none of the $2^{d-h-1}\binom{d}{h+1}$ partial assignments ρ, the event E_ρ holds. Taking a union bound,

$$\begin{aligned}\Pr[\text{P is not h-robust}] &\leqslant 2^{d-h-1}\binom{d}{h+1}2^{1-2^{h+1}} \\ &\leqslant 2^{d-h-2^{h+1}} d^{h+1} \\ &= o_d(1),\end{aligned}$$

for $h = \Theta(\log d)$. □

In this paper we analyze Goldreich's function $f_{P,G}$ where $G \in [n]^{n \times d}$ is a random bipartite graph. For Lemmas 4.10 and 4.13, the graph G should be a boundary expander, but there is a non-negligible (i.e. inverse polynomial in n) probability that G is not a good boundary expander. However, the following Lemma shows that always except with probability exponentially small in n the graph is an "imperfect" boundary expander. In the next subsection, we shall see that our results about drunk algorithms when G is a boundary expander also work when G is an imperfect boundary expander.

Definition 5.3 (Imperfect Expansion) *A graph* $G \in [n]^{n \times d}$ *is an* r_{bad}-imperfect (r, c)-boundary expander *if there exists a subset* $I_{bad} \subseteq R$ *of size* $|I_{bad}| \leqslant r_{bad}$ *such that* $G \setminus (I_{bad} \cup \Gamma(I_{bad}))$*, i.e. the graph obtained by removing vertices* I_{bad} *and* $\Gamma(I_{bad})$ *from* G*, is an* (r, c)*-boundary expander. We call* I_{bad} *an* extraneous set *of* G*.*

Lemma 5.4 *A random bipartite graph* $G \in [n]^{n \times d}$ *with* n *left nodes and* n *right nodes, and of right-degree* d*, is with probability* $1-(1/4)^{r_{bad}}$ *an* r_{bad}*-imperfect* (r, c)*-boundary expander for any* $c = d - \Omega(d)$*, provided* $r + r_{bad} \leqslant r_{max}(n, d, c)$*, where* $r_{max} = \Omega(n/d)$*.*

Proof. Let $c' = (c+d)/2$. Let I_{bad} be a largest set of right-nodes $I \subseteq R$ of size at most $r + r_{bad}$ such that $|\Gamma(I)| \leqslant c'|I|$. Remove I_{bad} and $\Gamma(I_{bad})$ from G. Then any set S of $\leqslant r + r_{bad} - |I_{bad}|$ right vertices in $G \setminus (I_{bad} \cup \Gamma(I_{bad}))$ has at least $c'|S|$ distinct neighbors, and hence has at least $(2c' - d)|S| = c|S|$ boundary neighbors (because every non-boundary neighbor is connected via $\geqslant 2$ edges to S). Thus in order to show G is an r_{bad}-imperfect (r, c)-boundary expander, we just need to show that $|I_{bad}| \leqslant r_{bad}$.

We note that the probability that a specific set I of i right nodes has $< c'i$ neighbors is at most

$$\binom{n}{\lfloor c'i \rfloor}\left(\frac{\lfloor c'i \rfloor}{n}\right)^{di} \leqslant \left(\frac{ne}{\lfloor c'i \rfloor}\right)^{\lfloor c'i \rfloor}\left(\frac{\lfloor c'i \rfloor}{n}\right)^{di}.$$

Thus the probability that $|I_{bad}| > r_{bad}$ is at most

$$\sum_{i=r_{bad}+1}^{r+r_{bad}} \binom{n}{i}\left(\frac{ne}{\lfloor c'i \rfloor}\right)^{\lfloor c'i \rfloor}\left(\frac{\lfloor c'i \rfloor}{n}\right)^{di} \leqslant \sum_{i=r_{bad}+1}^{r+r_{bad}} \left(\frac{ne}{i}\right)^{i}\left(\frac{ne}{c'i}\right)^{c'i}\left(\frac{c'i}{n}\right)^{di} = \sum_{i=r_{bad}+1}^{r+r_{bad}} \left(\left(\frac{i}{n}\right)^{d-c'-1} c'^{d-c'} e^{1+c'}\right)^{i}.$$

Let $a_i = (\frac{i}{n})^{d-c'-1} c'^{d-c'} e^{1+c'}$. Define

$$r_{\max} = \frac{n}{c'}(4c'e^{1+c'})^{-1/(d-c'-1)} = \Omega(n/d).$$

For $i \leqslant r_{\max}$, we have $a_i \leqslant 1/4$. Thus the above sum is at most $(1/4)^{r_{\text{bad}}}$. □

5.3 Coping with Imperfect Expansion

Lemma 5.5 *Let* $f = f_{G,P} : \{0,1\}^n \to \{0,1\}^m$ *be an instance of Goldreich's function. Let* $I \subseteq R$ *be a set of right-nodes in* G *and define* $\hat{f} = f_{(G\setminus I),P} : \{0,1\}^n \to \{0,1\}^{m-|I|}$.

Sample $x' \in \{0,1\}^n$ *uniformly at random and let* $b = f(x)$ *and* $\hat{b} = \hat{f}(x) = b_{R\setminus I}$. *Let* $\mathcal{A}$ *be a drunk backtracking algorithm for inverting* f *that returns the exact solution* x' *in time* $\leqslant$ maxtime *with probability* p. *Then there exists a drunk backtracking algorithm* $\mathcal{A}'$ *for inverting* $\hat{f}(x')$ *that given* $b_{R\setminus I}$ *with probability at least* $p2^{-|\Gamma(I)|}$ *returns the exact solution* x' *in time* $\leqslant$ maxtime $+ |\Gamma(I)|$.

A first attempt to prove Lemma 5.5 is to have $\mathcal{A}'$ convert the output $\hat{b} \in \{0,1\}^{m-|\Gamma(I)|}$ into a complete output $b \in \{0,1\}^m$ by guessing the output values b_I randomly. This guess would be correct with probability $2^{-|I|}$, and $\mathcal{A}'$ could then try to emulate the original algorithm $\mathcal{A}$ on the complete input b, by making the same decision that $\mathcal{A}$ would make at each node of the backtracking tree. The trouble with this approach is that when $\mathcal{A}$ reaches a node whose partial assignment is inconsistent with a bit in b_I, it can backtrack. $\mathcal{A}'$ is only allowed to backtrack from nodes which are inconsistent with bits in $\hat{b}$, and so $\mathcal{A}'$ may be forced to explore backtracking subtrees that $\mathcal{A}$ is allowed to skip. To fix this problem, we guess the input bits $x'_{\Gamma(I)}$ instead of guessing the output bits b_I.

Proof. $\mathcal{A}'$ will begin by assigning values to the bits $x_{\Gamma(I)}$ in a drunk (random) way. With probability $2^{-|\Gamma(I)|}$, the resulting partial assignment ρ agrees with x'. We are not interested in the behavior of $\mathcal{A}'$ if the partial assignment ρ does not agree with x. But if it agrees, then we continue $\mathcal{A}'$ as if $\mathcal{A}$ were running. However, sometimes $\mathcal{A}$ tries to assign a value to a variable in $\Gamma(I)$: In this case, $\mathcal{A}'$ has already assigned the correct value to that variable and $\mathcal{A}'$ does not need to assign a new value. Clearly, since ρ agrees with x', $\mathcal{A}$ will never backtrack because of the inconsistency of its current partial assignment with a bit b_i where $i \in I$. Rather, $\mathcal{A}$ will backtrack because of an inconsistency with a bit b_i where $i \in R\setminus I$; but, in this case $\mathcal{A}'$ will also backtrack. To summarize, conditioned on ρ agreeing with x', the running time of $\mathcal{A}'$ is at most $|\Gamma(I)|$ steps more than the running time of $\mathcal{A}$, and if $\mathcal{A}$ finds x' so does $\mathcal{A}'$. The statement of the lemma follows. □

5.4 Putting It All Together

Proof (Proof of Theorem 2.7). Given predicate P, we first check that P satisfies the test that implies the upper bound of Theorem 2.4. Next, we check that P is h-robust for $h = d/2 - \Omega(d)$. If both checks are satisfied, we say that the predicate has passed the test. By Lemma 5.2, the test satisfies properties (A), (B'), and (C).

Let $G \in [n]^{n\times d}$ be chosen uniformly at random. By Theorem 2.4 and Markov's inequality, $f_{P,G}$ has expected preimage size $\leqslant M^2$ with probability $\geqslant 1 - 1/M$ for $M = n^{O(1)}2^{2^{-\Theta(d)}n}$. Also let $c = d/2 + h$. Then $c = d - \Omega(d)$, and by Lemma 5.4, a random $G \in [n]^{n\times d}$ is an r_{bad}-imperfect (r, c)-boundary expander with probability $1 - 2^{-\Theta(n2^{-\Theta(d)})}$ for $r = \Theta(n/d)$, $r_{\text{bad}} = \Theta(n2^{-\Theta(d)})$, and $r + r_{\text{bad}} \leqslant r_{\max}(n, d, c)$, with extraneous set I_{bad}. To summarize, with probability $1 - 2^{-n2^{-\Theta(d)}+O(\log n)}$ over the choice of G, both G is an r_{bad}-imperfect (r,c)-boundary expander and $f_{P,G}$ has expected preimage size $\leqslant M^2$. We show that in this case, any drunk backtracking algorithm $\mathcal{A}$ cannot invert $\hat{f} = f_{P,G\setminus I_{\text{bad}}}$ with high probability efficiently.

Let $\mathcal{A}$ be a drunk backtracking algorithm. Choose $x \in \{0,1\}^n$ uniformly at random and let $\hat{b} = f_{P,G\setminus I_{\text{bad}}}(x)$. By Lemma 4.13, in finding an element in the preimage of $\hat{b}$ the clever version $\mathcal{C}(\mathcal{A})$ of $\mathcal{A}$ makes at least $\lfloor cr/4 \rfloor = \Omega(n)$ locally unforced assignments on the left-most branch of its backtracking tree. Let ρ be the partial assignment on the left-most branch after $\lfloor cr/4 \rfloor$ locally unforced assignments. By Lemma 4.13, ρ is locally consistent, and by Lemma 5.1, with probability $1 - 2^{-\Omega(n)}$, the partial assignment ρ does not agree with x. Assume ρ does not agree with x. Then

– either ρ is globally consistent in which case $\mathcal{C}(\mathcal{A})$ returns $\mathcal{C}(\mathcal{A})(\hat{b}) \neq x$ from its search of the backtracking tree rooted at ρ,
– or ρ is globally inconsistent, in which case, by Lemma 4.10, $\mathcal{C}(\mathcal{A})(\hat{b})$ spends $2^{(c/2-h)r/4-d}$ steps in searching the backtracking tree rooted at ρ. Noticing that $c/2 - h = \Omega(d)$, this amounts to $2^{\Theta(n)}$ steps.

In either case, $\mathcal{C}(\mathcal{A})$ and hence $\mathcal{A}$ does not return x as output in time better than $2^{\Theta(n)}$. So $\mathcal{A}$ outputs x in time better than $2^{\Theta(n)}$ with probability $\leqslant p = 2^{-\Theta(n)}$.

By Lemma 5.5, drunk backtracking algorithms for $f = f_{P,G}$ itself also cannot return x given $b = f(x)$ in time better than $2^{\Theta(n)}$ with probability better than $p2^{|\Gamma(I_{\text{bad}})|} = 2^{-\Theta(n)}$, because $|\Gamma(I_{\text{bad}})| \leqslant d|I_{\text{bad}}| = \Theta(n2^{-\Theta(d)})$.

By Lemma 3.1, drunk backtracking algorithms cannot return any element in the preimage $f^{-1}_{P,G}(b)$ in time better than $2^{\Theta(n)}$ with probability better than $2\sqrt{M^2 2^{-\Theta(n)}}$. This probability is at most $2^{-C(d)n}$ where $C(d)$ only depends on d and is positive for large enough d. This completes the proof. □

6 Myopic Backtracking Algorithms

The goal of this section is to prove Theorem 2.10.

6.1 Choice of Predicate and Graph

To prove the security of Goldreich's function against drunk backtracking algorithms, we assumed the predicate P was h-robust, that the graph G was an (r,c)-expander, and that G and P were such that $f_{P,G}$ had small preimages. In the case of myopic backtracking algorithms, we add one more condition on P and one more condition on G.

Definition 6.1 (Balanced Predicate) *For a predicate $P : \{0,1\}^d \to \{0,1\}$, real number $\epsilon_{\text{bal}} \in [0,1/2)$, and integer $h \in [0, d-1]$, we say predicate P is* $(h, \epsilon_{\text{bal}})$-balanced *if after fixing all but $h+1$ variables,*

$$|\Pr[P(x_1,\ldots,x_d) = 0|\ \textit{fixed variables}] - \tfrac{1}{2}| \leqslant \epsilon_{\text{bal}}.$$

For example, the predicate $P_{2,\wedge} = x_1 \oplus \cdots \oplus x_{d-2} \oplus (x_{d-1} \wedge x_d)$ is $(2,0)$-balanced and $(1,\frac{1}{4})$-balanced. The predicate that sums all its inputs modulo 2 is $(0,0)$-balanced. A predicate is h-robust iff there is some $\epsilon_{\text{bal}} \in [0,1)$ such that P is $(h,\epsilon_{\text{bal}})$-balanced.

Lemma 6.2 *A random predicate on d variables is $(\Theta(\log \frac{d}{\epsilon_{\text{bal}}}), \epsilon_{\text{bal}})$-balanced with probability $1 - \exp[-\text{poly}(d/\epsilon_{\text{bal}})]$. Predicates of the form $P_{h,Q} = x_1 \oplus \cdots \oplus x_{d-h} \oplus Q(x_{d-h+1},\ldots,x_d)$ are $(h,0)$-balanced.*

Proof. Let ρ be any partial assignment which fixes all but $h+1$ variables. There are 2^{h+1} inputs consistent with ρ: call them $x_1^\rho, \ldots, x_{2^{h+1}}^\rho$. Let E_ρ be the event that P is not balanced under the partial assignment ρ: $E_\rho = \{|\#\{i : P(x_i^\rho) = 1\} - 2^h| > 2^{h+1}\epsilon_{\text{bal}}\}$.

P is balanced if for none of the $2^{d-h-1}\binom{d}{h+1}$ partial assignments ρ, the event E_ρ holds. By a Chernoff bound, $\Pr[E_\rho] \leqslant 2e^{-\epsilon_{\text{bal}}^2 2^{h+2}}$. Taking a union bound,

$$\begin{aligned}\Pr[P \text{ is not } (h, \epsilon_{\text{bal}})\text{-balanced}] &\leqslant 2^{d-h-1}\binom{d}{h+1}2e^{-\epsilon_{\text{bal}}^2 2^{h+2}} \\ &\leqslant 2^{d-h}d^{h+1}e^{-\epsilon_{\text{bal}}^2 2^{h+2}} \\ &= \exp[(h+1)\ln d + (d-h)\ln 2 - \epsilon_{\text{bal}}^2 2^{h+2}].\end{aligned}$$

Finally, to show the desired result, take $h = \Theta(\log \frac{d}{\epsilon_{\text{bal}}})$.

To see that the predicate $P_{h,Q}(x_1, \ldots, x_d) = x_1 \oplus \cdots \oplus x_{d-h} \oplus Q(x_{d-h+1}, \ldots, x_d)$ is $(h, 0)$-balanced, notice that any subset of $h+1$ variables includes at least one of the variables $x_1, \ldots, x_{d-h}$. □

Lemma 6.3 *A bipartite graph $G \in [n]^{n \times d}$ chosen uniformly at random from $[n]^{n\times d}$, with n left nodes, n right nodes, and of right-degree d has with probability $\geqslant 1 - 2^{-n_{\text{bad}}}$ at most n_{bad} left nodes of degree $> d_{\text{left}}$, provided $d_{\text{left}} \geqslant 2ed$ and $n_{\text{bad}} \geqslant 2en2^{-d}$.*

Proof. The probability that there are $> n_{\text{bad}}$ left vertices of degree $> d_{\text{left}}$ is at most the probability that there exists a set S of n_{bad} left vertices such that at least $n_{\text{bad}}d_{\text{left}}$ of the edges of the graph have an endpoint in S. For each set S of n_{bad} left vertices, this happens with probability at most

$$\binom{nd}{n_{\text{bad}}d_{\text{left}}}\left(\frac{n_{\text{bad}}}{n}\right)^{n_{\text{bad}}d_{\text{left}}} \leqslant \left(\frac{nde}{n_{\text{bad}}d_{\text{left}}}\right)^{n_{\text{bad}}d_{\text{left}}}\left(\frac{n_{\text{bad}}}{n}\right)^{n_{\text{bad}}d_{\text{left}}} \leqslant 2^{-dn_{\text{bad}}},$$

where the last inequality is true provided $d_{\text{left}} \geqslant 2de$. Now by a union bound, the probability that such an S exists is at most

$$\binom{n}{n_{\text{bad}}}2^{-dn_{\text{bad}}} \leqslant \left(\frac{ne}{n_{\text{bad}}}\right)^{n_{\text{bad}}}2^{-dn_{\text{bad}}} \leqslant 2^{-n_{\text{bad}}}$$

provided $n_{\text{bad}} \geqslant 2ne2^{-d}$. □

6.2 Probability of a Correct Guess

Lemma 6.4 (Main Myopic Lemma) *Let $\mathcal{A}$ be an (s, r)-myopic backtracking algorithm for inverting Goldreich's function $f_{P,G}$, where $s = 2^{-o(d)}n$. Assume that we are guaranteed that $\mathcal{A}$ makes at least s assignments on the left-most branch of its backtracking tree before it stops searching that path. Let P be any $(h, \epsilon_{\text{bal}})$-balanced predicate where $\epsilon_{\text{bal}} = 2^{-\Omega(d)}$, and let $G \in [n]^{m\times d}$ be any (r, c)-boundary expander with $m = O(n)$ where all but $n_{\text{bad}} = 2^{-\Omega(d)}n$ of the left-nodes have degree $O(d)$.*

Choose $x' \in \{0,1\}^n$ uniformly at random and let $b = f_{P,G}(x')$. When b is given as input to $\mathcal{A}$, let $\rho^s(b)$ be the resulting partial assignment on the left-most branch after s assignments. The probability, over the randomness of $x' \in \{0,1\}^n$, that $\rho^s(b)$ can be extended to x' is at most $2^{-2^{-o(d)}n}$.

Lemma 6.5 *Assume* G *is an* (r,c)*-boundary expander of right-degree* d*, with at most* n_{bad} *left-nodes of degree bigger than* d_{left}*. Let* $h \geqslant 0$ *be a real number and assume* $c > h+1$*. Let* W *be a set of left-nodes of* G*. Then there exists* $U \subseteq W$ *such that*

$$|U| \geqslant \frac{|W| - n_{\text{bad}}}{2d_{\text{left}}d}$$

and for every $A \subseteq R$ *with* $|A| \leqslant r$*, there is some* $i \in A$ *such that* $|\partial_A i \setminus U| > h$*.*

Proof. Let $\hat{c} = \lceil c-h-1 \rceil$. Construct U using the following algorithm:

```
– $U \leftarrow \varnothing$.
– For every $i \in L$, set $n_i \leftarrow \begin{cases} \hat{c} & \text{if } i \in W \text{ and } i \text{ has degree at most } d_{\text{left}}; \\ 0 & \text{otherwise.} \end{cases}$
– The following invariant holds every time the following **while** loop checks its loop condition: every right-node connected to a left-node $j$ has at most $\hat{c} - n_j$ adjacent left-nodes in $U$.
– **while** $\exists i,\ n_i > 0$,
    • $U \leftarrow U \cup \{i\}$.
    • $n_i \leftarrow 0$.
    • For every $j \in L$ distinct from $i$, if $i$ and $j$ have a common neighbor, then $n_j \leftarrow \max\{0, n_j - 1\}$.
```

In the beginning, $\sum_i n_i \geqslant \hat{c}(|W| - n_{\text{bad}})$, and in the end, $\sum_i n_i = 0$. At each step, $\sum_i n_i$ decreases by at most $\hat{c} + d_{\text{left}}(d-1)$. Therefore the number of steps we took is

$$|U| \geqslant \frac{\hat{c}(|W| - n_{\text{bad}})}{\hat{c} + d_{\text{left}}(d-1)} \geqslant \frac{|W| - n_{\text{bad}}}{2d_{\text{left}}d}.$$

By the loop invariant, every right-node has at most $\hat{c}$ adjacent left-nodes in U. Let $A \subseteq R$ have size $|A| \leqslant r$. Then by the expansion of G, there is some $i \in A$ such that $|\partial_A i| \geqslant c$. It follows that $|\partial_A i \setminus U| \geqslant c - \hat{c} > h$. □

Proof (of Lemma 6.4). As the myopic algorithm $\mathcal{A}$ runs, it may query bits of $b = f(x')$ in order to decide which assignment to try next. Let $T(b) \subseteq [m]$ be the set of indices of bits of b that $\mathcal{A}$ queries after following the leftmost branch for s steps. Since $\mathcal{A}$ is (s,r)-myopic, we know $|T(b)| \leqslant r$. The decisions of $\mathcal{A}$ are based only on the bits in $T(b)$, so whenever $b'_{T(b)} = b_{T(b)}$, it must be that $\rho^s(b') = \rho^s(b)$ and $T(b') = T(b)$.

For any $\hat{b} \in \{0,1\}^n$, define the set

$$E_{\hat{b}} = \{x' \in \{0,1\}^n : f(x')_{T(\hat{b})} = \hat{b}_{T(\hat{b})}\}.$$

We begin by showing that the sets $\{E_{\hat{b}} : \hat{b} \in \{0,1\}^n\}$ form a partition of $\{0,1\}^n$. These sets cover all of $\{0,1\}^n$ because $x' \in E_{f(x')}$ for every $x' \in \{0,1\}^n$. Now assume, for $\hat{b}, \hat{b'} \in \{0,1\}^n$, the sets $E_{\hat{b}}$ and $E_{\hat{b'}}$ share a string x'. Then $f(x')_{T(\hat{b})} = \hat{b}_{T(\hat{b})}$, so $T(f(x')) = T(\hat{b})$ and similarly $T(f(x')) = T(\hat{b'})$. Thus, $T(\hat{b}) = T(\hat{b'})$, and $\hat{b}_{T(\hat{b})} = f(x')_{T(\hat{b})} = \hat{b'}_{T(\hat{b'})}$. This means that any two intersecting sets $E_{\hat{b}}$ and $E_{\hat{b'}}$ are equal.

Since the sets $E_{\hat{b}}$ partition $\{0,1\}^n$, we can prove the Lemma by showing that for every $\hat{b}$, the probability that ρ^s can be extended to x', conditioned on event $x' \in E_{\hat{b}}$, is at most $2^{-2^{-o(d)}n}$.

Therefore from now on we fix $\hat{b}$. Conditioning on the event $x' \in E_{\hat{b}}$ fixes ρ^s. By Lemma 6.5 with $W = \text{Vars}(\rho^s)$, there exists a set of input nodes $U \subseteq \text{Vars}(\rho^s)$ of size $(s - n_{\text{bad}})/(2d\Theta(d)) = 2^{-o(d)}n$ such that every subset of $T(b)$ has boundary expansion $> h$ outside U. We know that

$$\Pr[\rho^s \text{ can be extended to } x' | x' \in E_{\hat{b}}] \leqslant \Pr[x'_U = \rho^s_U | x' \in E_{\hat{b}}],$$

so it suffices to show

$$\Pr[x_U = y | x' \in E_{\hat{b}}] = 2^{-2^{-o(d)}n}, \tag{6}$$

for $y = \rho^s_U$.

Here is a two-sentence overview of the proof of (6) for any $y \in \{0,1\}^{|U|}$. We first show that x_U has little influence on the distribution of $b_{T(\hat{b})}$. Then by Bayes' rule, we conclude that the bits $b_{T(\hat{b})}$ do not contain much information about x'_U.

Order the nodes in $T(\hat{b})$ as $v_1, v_2, \dots, v_{|T(\hat{b})|}$ such that for every $1 \leqslant i \leqslant |T(\hat{b})|$, we have $|\Gamma(T_i) \setminus (\Gamma(T_{i-1}) \cup U)| \geqslant h+1$ for $T_i = \{v_1, \dots, v_i\}$. This ordering is possible because every subset of $T(\hat{b})$ has a node with boundary $> h$ outside U. For any $y \in \{0,1\}^{|U|}$, we have

$$\begin{aligned}\Pr[x' \in E_{\hat{b}} | x_U = y] &= \prod_{i=1}^{|T(\hat{b})|} \Pr[b_{v_i} = \hat{b}_{v_i} | b_{T_{i-1}} = \hat{b}_{T_{i-1}}, x_U = y] \\ &\in [(\tfrac{1}{2} - \epsilon_{\mathrm{bal}})^{|T(\hat{b})|}, (\tfrac{1}{2} + \epsilon_{\mathrm{bal}})^{|T(\hat{b})|}],\end{aligned}$$

since P is $(h, \epsilon_{\mathrm{bal}})$-balanced and since for every $1 \leqslant i \leqslant |T(\hat{b})|$, v_i has at least $h+1$ neighbors outside $\Gamma(T_{i-1})$ and U. Using Bayes' rule, for any $y, y' \in \{0,1\}^{|U|}$,

$$\begin{aligned}\frac{\Pr[x'_U = y' | x' \in E_{\hat{b}}]}{\Pr[x'_U = y | x' \in E_{\hat{b}}]} &= \frac{\Pr[x' \in E_{\hat{b}} | x'_U = y'] \Pr[x'_U = y']}{\Pr[x' \in E_{\hat{b}} | x'_U = y] \Pr[x'_U = y]} \\ &= \frac{\Pr[x' \in E_{\hat{b}} | x'_U = y']}{\Pr[x' \in E_{\hat{b}} | x'_U = y]} \\ &\geqslant \left(\frac{1 - 2\epsilon_{\mathrm{bal}}}{1 + 2\epsilon_{\mathrm{bal}}}\right)^{|T(\hat{b})|}.\end{aligned}$$

Fixing y and summing the above inequality over all $y' \in \{0,1\}^{|U|}$, we get

$$\frac{1}{\Pr[x'_U = y | x' \in E_{\hat{b}}]} \geqslant 2^{|U|} \left(\frac{1 - 2\epsilon_{\mathrm{bal}}}{1 + 2\epsilon_{\mathrm{bal}}}\right)^{|T(\hat{b})|}.$$

Since $|U| = 2^{-o(d)}n$, $\epsilon_{\mathrm{bal}} = 2^{-\Omega(d)}$ and $|T(\hat{b})| \leqslant m = O(n)$, Equation (6) follows. □

6.3 Clever Myopic Algorithms

We will need a stronger version of Lemma 4.15.

Lemma 6.6 *Let $J \subseteq L$ have size $|J| \leqslant cr/4$. Then there exists a closure C for J such that $|C| \leqslant 2c^{-1}|J|$ and $|\partial C \setminus J| \leqslant c|C|/2$.*

Furthermore, if C' is any set such that $|C'| \leqslant 2c^{-1}|J|$ and $|\partial C' \setminus J| \leqslant c|C'|/2$, then we can pick C to be a superset of C'.

Proof. The proof is the same as the proof of Lemma 4.15, except for the following change. Instead of saying "Let C be a largest nonexpanding set with $\leqslant r/2$ vertices", the proof should say "Let C be a largest superset of C' which is nonexpanding and has $\leqslant r/2$ vertices." We know there is such a set because C' itself is a superset of C' which is nonexpanding and has $\leqslant r/2$ vertices. □

Consider a myopic backtracking algorithm $\mathcal{A}$ and the clever version $\mathcal{C}(\mathcal{A})$. In order to decide which assignment to make next, $\mathcal{C}(\mathcal{A})$ queries the same bits of b that $\mathcal{A}$ would, but it also checks at every step whether the partial assignment is locally consistent. Since $\mathcal{C}(\mathcal{A})$ makes this additional kind of query, it is not clear whether $\mathcal{C}(\mathcal{A})$ is myopic. However, as the following lemma shows, it is possible to create a myopic algorithm which behaves in the same way as $\mathcal{C}(\mathcal{A})$ by reading a limited number of additional bits of b.

Lemma 6.7 *Let $\mathcal{A}$ be an (s,t)-myopic backtracking algorithm for inverting Goldreich's function $f_{P,G}$. If G is an (r,c)-boundary expander, $s \leqslant cr/4$ and P is a $(c/2-1)$-robust predicate, then there is an $(s, t+2c^{-1}s)$-myopic backtracking algorithm $\mathcal{A}'$ which has the same backtracking tree as $\mathcal{C}(\mathcal{A})$.*

Proof. During the task of finding $x \in f^{-1}_{P,G}(b)$, algorithm $\mathcal{A}'$ will maintain a set $C \subseteq R$ of right-nodes of G. Whenever $\mathcal{A}'$ adds a new node i to C, it will also query the corresponding bit b_i. When $\mathcal{A}'$ finishes exploring a subtree and consequently forgets some bits of b, it removes the corresponding nodes from the set C: so $\mathcal{A}'$ always knows every bit b_i for $i \in C$. The algorithm will use this knowledge of b in order to emulate the clever decisions made by $\mathcal{C}(\mathcal{A})$.

$\mathcal{A}'$ will maintain the following invariant at every point in its execution where it has assigned values to less than s variables. Let ρ be the current partial assignment, let $x_j \leftarrow a$ be the next assignment that $\mathcal{A}$ would make at this point, and let $J = \mathrm{Vars}(\rho) \cup \{j\}$. Then C is a closure for J, $|C| \leqslant 2c^{-1}|J|$, and $|\partial C \setminus J| \leqslant c|C|/2$. Since $s \leqslant cr/4$, Lemma 6.6 ensures that $\mathcal{A}'$ can maintain this invariant.

Now, whenever $\mathcal{C}(\mathcal{A})$ makes a new assignment, it makes the same assignment $x_j \leftarrow a$ that $\mathcal{A}$ would make, unless $x_j \leftarrow 1-a$ is locally forced. Therefore, to see that $\mathcal{A}'$ can emulate $\mathcal{C}(\mathcal{A})$, all that remains is to show that the bits b_i for $i \in C$ always provide enough information to determine whether any partial assignment to the variables in J is locally consistent. Indeed, if a partial assignment to J is consistent with C, then it is locally consistent by definition, and conversely, if it is locally consistent, then by Lemma 4.16 it is consistent with C. □

6.4 Coping with Imperfect Expansion

So far, when studying the behavior of myopic backtracking algorithms, we have assumed that the graph used to construct Goldreich's function is an (r,c)-boundary expander. However, as discussed in Section 5.2, the probability that a graph is not a boundary expander is non-negligable. Here we show that the results in this section also apply to imperfect boundary expanders, analagously to Lemma 5.5.

Lemma 6.8 *Let $f = f_{G,P} : \{0,1\}^n \to \{0,1\}^m$ be an instance of Goldreich's function. Let $I \subseteq R$ be a set of right-nodes in G and define $\hat{f} = f_{(G\setminus I),P} : \{0,1\}^n \to \{0,1\}^{m-|I|}$.*

Sample $x' \in \{0,1\}^n$ uniformly at random and let $b = f(x)$ and $\hat{b} = \hat{f}(x) = b_{R\setminus I}$. Let $\mathcal{A}$ be an (s,t)-myopic backtracking algorithm for inverting f that returns the exact solution x' in time $\leqslant \mathrm{maxtime}$ with probability p. Then there exists an (s,t)-myopic backtracking algorithm $\mathcal{A}'$ for inverting $\hat{f}(x')$ that given $b_{R\setminus I}$ with probability at least $p2^{-|\Gamma(I)|}$ returns the exact solution x' in time $\leqslant \mathrm{maxtime} + |\Gamma(I)|$.

Proof. The proof of Lemma 5.5 gives us a randomized backtracking algorithm $\mathcal{A}'$ which with probability $\geqslant p2^{-|\Gamma(I)|}$ returns x' in time $\leqslant \mathrm{maxtime} + |\Gamma(I)|$. This algorithm is (s,t)-myopic, since while it is guessing the bits $x_{\Gamma(I)}$ it does not look at b at all, and the rest of its decisions are made according to the myopic algorithm $\mathcal{A}$. □

6.5 Putting It All Together

Proof (Proof of Theorem 2.10). The proof is the same as the proof of Theorem 2.7 in Section 5.4, with the following changes.

- We test the predicate P with the same test as in Theorem 2.7, but we additionally test that P is $(h, \epsilon_{\mathrm{bal}})$-balanced for $h = d/2 - \Omega(d)$ and $\epsilon_{\mathrm{bal}} = 2^{-\Omega(d)}$. If both the old and new tests are satisfied, we say that P has passed the test. By Lemma 6.2, this new test satisfies properties (A), (B'), and (C).
- We use Lemma 6.3 to show that a random G has with probability $1 - 2^{-2^{-\Theta(d)}n}$ no more than $n_{\mathrm{bad}} = 2en2^{-d}$ left vertices of degree $> 2ed$.
- Rather than assuming $\mathcal{A}$ is a drunk algorithm, we assume $\mathcal{A}$ is (s,t)-myopic, where $s = 2^{-o(d)}n$ and $t = r/2 = \Omega(n/d)$. Let $s' = \min\{s, \lfloor cr/4 \rfloor\} = 2^{-o(d)}n$. Then $\mathcal{A}$ is (s', t)-myopic. By Lemma 6.7, the clever version $\mathcal{C}(\mathcal{A})$ is $(s', t + r/2)$-myopic, so we can apply Lemma 6.4 to $\mathcal{C}(\mathcal{A})$. As for the success probability of algorithm $\mathcal{A}$ on $f_{P,G\setminus I_{\mathrm{bad}}}$ in finding x itself, instead of the upper bound of $p = 2^{-\Theta(n)}$, we get the upper bound of $p = 2^{-2^{-o(d)}n}$.
- We use Lemma 6.8 where the proof of Theorem 2.10 uses Lemma 5.5. This shows a drunk backtracking algorithm on $f_{P,G}$ has success probability of at most $2^{-2^{-o(d)}n}$ in finding x itself. This shows that the success probability of finding any element in the preimage is at most $2\sqrt{M^2 2^{-2^{-o(d)}n}}$, and this completes the proof.

□

7 Myopic and Drunk DPLL Backtracking Algorithms

Here we prove Theorem 2.13.

7.1 Simulating DPLL Algorithms

Let us forget myopic and drunk algorithms for a moment, and compare backtracking algorithms as described by Definition 2.5 to DPLL backtracking algorithms described by Definition 2.12. Both kinds of algorithm have the option at every step of taking a free variable x_j and a bit a and trying first the assignment $x_j \leftarrow a$ and then, if no solution was found on that branch, the assignment $x_j \leftarrow 1 - a$. A DPLL algorithm is also able to make a DPLL assignment $x_j \leftarrow a$ and skip the alternative assignment $x_j \leftarrow 1 - a$. The following lemma shows that this additional ability does not help a DPLL algorithm, except to decrease the number of nodes it must explore by a factor of at most 2^d.

Lemma 7.1 *Let $f_{P,G}$ be an instance of Goldreich's function, where P is any d-ary predicate and G is any graph. For any DPLL backtracking algorithm $\mathcal{D}$, there is a (non-DPLL) backtracking algorithm $\mathcal{S}(\mathcal{D})$ which simulates $\mathcal{D}$: When algorithms $\mathcal{D}$ and $\mathcal{S}(\mathcal{D})$ are given b and asked to find some $x \in f_{P,G}^{-1}(b)$, then they both return the same result, and the backtracking tree of $\mathcal{S}(\mathcal{D})$ contains at most 2^d times as many nodes as that of $\mathcal{D}$.*

Proof. Whenever $\mathcal{D}$ makes a non-DPLL assignment $x_j \leftarrow a$, the simulation $\mathcal{S}(\mathcal{D})$ makes the same assignment. However, if $\mathcal{D}$ makes a DPLL assignment $x_j \leftarrow a$, then the behavior of $\mathcal{S}(\mathcal{D})$ depends on whether the assignment is a unit clause or a pure literal.

If $x_j \leftarrow a$ is a pure literal, $\mathcal{S}(\mathcal{D})$ postpones making this assignment by adding it to a "to-do list" of pure literal assignments. The following invariant is always maintained:

> *If the union of the current partial assignment of $\mathcal{S}(\mathcal{D})$ together with its to-do list contradicts any equation $b_i = f_{P,G}(x)_i = P(x_{G_{i,1}}, \ldots, x_{G_{i,d}})$, then the current partial assignment without the to-do list contradicts that same equation.*

This means that whenever $\mathcal{D}$ returns back from a recursion because some bit b_i has been contradicted, $\mathcal{S}(\mathcal{D})$ can also return back in $\leqslant 2^d - 1$ steps by exhaustively assigning values to all of the variables $x_{G_{i,1}}, \ldots, x_{G_{i,d}}$ which are still free. (Assuming P is non-trivial, it has at most $d-1$ free variables at this point, and $2^d - 1$ is the number of nodes in a complete binary tree of depth $d-1$.)

The other possible action $\mathcal{D}$ might take is to assign a unit clause $x_j \leftarrow a$. This means that there is some output bit i such that the equation $b_i = P(x_{G_{i,1}}, \ldots, x_{G_{i,d}})$ implies $x_j = a$. In this case, $\mathcal{S}(\mathcal{D})$ first makes the assignment $x_j \leftarrow 1 - a$. Together with the assignments in the to-do list, this contradicts the equation $b_i = P(x_{G_{i,1}}, \ldots, x_{G_{i,d}})$, so by the invariant, $\mathcal{S}(\mathcal{D})$ can exhaust the branch in $\leqslant 2^d - 1$ steps. $\mathcal{S}(\mathcal{D})$ is then free to make the DPLL assignment $x_j \leftarrow a$.

If $\mathcal{D}$ succeeds in finding a complete input $x \in f^{-1}(b)$, then $\mathcal{S}(\mathcal{D})$ proceeds to make all the pure literal assignments in its to-do list, and arrives at the same complete assignment x. $\square$

7.2 Drunk DPLL Algorithms

Lemma 7.2 . *If $\mathcal{D}$ is a drunk DPLL backtracking algorithm, then there is a non-DPLL backtracking algorithm $\mathcal{S}'(\mathcal{D})$ that relates to $\mathcal{D}$ in the same way as $\mathcal{S}(\mathcal{D})$ in Lemma 7.1, except that $\mathcal{S}'(\mathcal{D})$ is drunk, and also might return a different $x \in f^{-1}(b)$ from the one $\mathcal{S}(\mathcal{D})$ returns.*

Proof. The simulation $\mathcal{S}(\mathcal{D})$ from the proof of Lemma 7.1 makes four kinds of assignment. The first kind copies the non-DPLL assignments made by $\mathcal{D}$. These are already drunk, so $\mathcal{S}'(\mathcal{D})$ can behave the same way. The second kind of assignment is made when $\mathcal{S}(\mathcal{D})$ is exhaustively assiging values some set of $\leqslant d-1$ variables. In this case, we lose nothing by having $\mathcal{S}'(\mathcal{D})$ behave the same way except to try each pair of assignments $x_j \leftarrow 0$ and $x_j \leftarrow 1$ in a random order. The third kind of assignment is when $\mathcal{D}$ has assigned a unit clause $x_j \leftarrow a$. In this case, $\mathcal{S}(\mathcal{D})$ first tries the incorrect assignment $x_j \leftarrow 1-a$, but we lose nothing by having $\mathcal{S}'(\mathcal{D})$ try the assignments $x_j \leftarrow a$ and $x_j \leftarrow 1-a$ in random order.

The fourth and final kind of assignment $\mathcal{S}(\mathcal{D})$ makes is when $\mathcal{D}$ has found a solution $x \in f^{-1}(b)$, and $\mathcal{S}(\mathcal{D})$ proceeds to make all the assignments in its to-do list of pure literal assignments, in the same order they were added to the list. In this case, $\mathcal{S}'(\mathcal{D})$ will make the assignments in *reverse* order, and will chose the bit to assign randomly in each case. If $x_j \leftarrow a$ is a pure literal with respect to a partial assignment ρ, then it is still a pure literal with respect to any ρ' which extends ρ. Therefore, if $\mathcal{S}'(\mathcal{D})$ makes assignments in its to-do list in reverse order, then at each step, the remaining assignments still form a sequence of pure literal assignments. Now, if the last assignment on the to-do list is $x_j \leftarrow a$ and $\mathcal{S}'(\mathcal{D})$ drunkenly assigns $x_j \leftarrow 1-a$ instead, let x' be the full assignment we get by applying the remaining pure literals from the to-do list. Now, either $f(x') = b$, in which case we continue making assignments in the to-do list in reverse order, or for some i, $f(x')_i \neq b_i$, and since the remaining assignments are all pure literals, there is no way to satisfy $b_i = P(x_{G_{i,1}}, \ldots, x_{G_{i,d}})$ given the values already assigned. In the latter case, we exhaust all possible assignments to those variables in $\leqslant 2^d - 1$ steps, and continue with the correct assignment $x_j \leftarrow a$. $\square$

Proof (Proof of Theorem 2.13 for DPLL drunk backtracking algorithms). The theorem in the case of DPLL drunk backtracking algorithms follows immediately from Theorem 2.7 and Lemma 7.2. □

7.3 Myopic DPLL Algorithms

Definition 7.3 (Nonmonotone Predicate) *Let $0 \leqslant h < d$ be an integer. The predicate $P : \{0,1\}^d \to \{0,1\}$ is* h-nonmonotone *iff after any partial assignment of values to the input variables of P that lets $> h$ of the input variables be free, the resulting restricted predicate is not constant and has the following additional property. Pick any input bit x_i among the $\geqslant h+1$ unrestricted inputs. Then either the restricted predicate does not depend on x_i, or the restricted predicate is not monotone in x_i.*

Lemma 7.4 *The predicate $P = x_1 \oplus \cdots \oplus x_{d-h} \oplus Q(x_{d-h+1}, \ldots, x_d)$ is $(h+1)$-nonmonotone. A random predicate P on d variables is $\Theta(\log d)$-nonomonotone with probability $1 - o_d(d)$.*

Proof. By Lemma 5.2, we may assume that P is h-robust, and so only the "additional property" part of Definition 7.3 remains to be shown.

If we fix all but $h+1$ input bits and select an input bit x_i among those $h+1$, then the probability that a random d-ary predicate is monotone nondecreasing in x_i is exactly $(3/4)^{2^h}$. Taking a union bound, the probability that for any choice of $h+1$ variables, together with an index i among those $h+1$ and a way of fixing the remaining $d-h-1$ variables, the predicate becomes monotone in x_i is at most

$$2\binom{d}{h+1}2^{d-h-1}(h+1)(3/4)^{2^h},$$

which is $\exp(-\mathrm{poly}(d))$ if we take $h = \Theta(\log d)$.

Now, consider a predicate $P_{h,Q} = x_1 \oplus \cdots \oplus x_{d-h} \oplus Q(x_{d-h+1}, \ldots, x_d)$ where Q is any h-ary predicate. Consider any subset $S \subseteq [d]$ of $h+2$ variable indices, together with an index $i \subseteq S$ and a partial assignment ρ that fixes $x_{[d]\setminus S}$. We are to show that after the assignments in ρ are fixed, P either does not depend on x_i or is not monotone in x_i. We have two cases.

Case 1. $1 \leqslant i \leqslant d-h$. Then x_i is not one of the inputs to the predicate Q. Since $|S| \geqslant h+2$, there is at least one other unrestricted input x_j that is not an input to Q, so our predicate has the form $x_i \oplus x_j \oplus P'(x_{[d]\setminus\{i,j\}})$. This is not monotone in x_i.

Case 2. $d-h+1 \leqslant i \leqslant d$. In this case, x_i is an input to Q. Since $|S| \geqslant h+1$, there is at least one unrestricted input x_j that is not an input to Q, so our predicate has the form $x_j \oplus P'(x_i, x_{[d]\setminus\{i,j\}})$ where P' is a $(d-1)$-ary predicate. We may assume that after assignments in ρ are fixed, P' does not ignore x_i, so there is some value $\hat{x}$ for $x_{[d]\setminus\{i,j\}}$ which is consistent with ρ and such that $P'(0,\hat{x}) \neq P'(1,\hat{x})$. For different values of x_j, then, P increases or decreases with x_i, so P is not monotone in x_i. □

Lemma 7.5 *Let $f_{G,P}$ be an instance of Goldreich's function for graph G and an h-nonmonotonoe predicate P. Consider the execution of a DPLL backtracking algorithm which is searching for $x \in f^{-1}(b)$, where the current partial assignment is ρ. Let $I \subseteq R$ be a set of indices such that every node in $R \setminus I$ has at least $h+1$ distinct neighbors which are not in $\mathrm{Vars}(\rho)$. Then without reading any bits b_i for $i \notin I$, it is possible to know all the DPLL assignments which can be made starting from ρ.*

Proof. In order to know which unit clause assignments can be made, iterate through the bits b_i for $i \in I$. In each case, check whether, given ρ, the equation $b_i = P(x_{G_{i,1}}, \dots, x_{G_{i,d}})$ together with ρ implies any variable x_j has a particular value a. If any of these equations force x_j to take a value a, then $x_j \leftarrow a$ is a unit clause.

Finding all the pure literal assignments requires a bit more work. For every possible assignment $x_j \leftarrow a$, check whether it is a pure literal assignment as follows. First, iterate through all output bits b_i – even the ones we are not allowed to read – and in each case, consider the equation $b_i = P(x_{G_{i,1}}, \dots, x_{G_{i,d}})$. Even if we don't know b_i, we can still check whether, after fixing the variables in ρ, the truth of the equation can ever depend on the value of x_j (equivalently, whether P ignores the input x_j once ρ is fixed). If the truth never depends on x_j, we call output b_i *passive* with respect to x_j. Now, iterate through the bits b_i for $i \in I$, and note which of the equations $b_i = P(x_{G_{i,1}}, \dots, x_{G_{i,d}})$ can never be changed to false by setting $x_j = a$. Call these output bits *monotone toward* $x_j = a$. If x_j is only connected to outputs which are passive with respect to x_j and outputs $i \in I$ which are monotone toward $x_j = a$, then $x_j \leftarrow a$ is a pure literal assignment.

What is left is to show that the above procedure finds every unit clause and pure literal assignment.

If $x_j \leftarrow a$ is a unit clause assignment, then there must be some output bit b_i such that the equation $b_i = P(x_{G_{i,1}}, \dots, x_{G_{i,d}})$ implies $x_j = a$. Since P is h-nonmonotone, this implies that output i must be connected to $\leqslant h$ inputs not in Vars(ρ), so $i \in I$, and so the above procedure finds the assignment.

If $x_j \leftarrow a$ is a pure literal, then given ρ, every output bit b_i connected to x_j is either passive with respect to x_j or is monotone toward $x_j = a$. Now, if an output bit b_i is not passive with respect to x_j but is monotone toward $x_j = a$, then since P is h-nonmonotone, b_i must be connected to $\leqslant h$ inputs which are not in Vars(ρ). Therefore, the above procedure will find $x_j \leftarrow a$. □

Proof (Proof of Theorem 2.13 for DPLL myopic backtracking algorithms). Given a predicate P, we first check that P satisfies the test of Theorem 2.10. We then check that P is h-nonmonotone, for $h = d/2 - \Omega(d)$. If both checks are satisfied, we say that the predicate has passed the test. By Lemma 7.4, the test satisfies properties (A), (B'), and (C).

Let $t_{\mathrm{non-DPLL}}$ be the value given by Theorem 2.10 such that Goldreich's function is secure against $(s, t_{\mathrm{non-DPLL}})$-myopic bactracking algorithms. The proof of Theorem 2.10 guarantees that we can have $t_{\mathrm{non-DPLL}} = \Omega(n/d)$.

Let $G \in [n]^{n\times d}$ be chosen uniformly at random. Let $c = d/2+h$. Then $c = d-\Omega(d)$, and by Lemma 5.4, G is an r_{bad}-imperfect (r, c)-boundary expander with probability $\geqslant 1-2^{-\Theta(n/d)}$ for $r = \Omega(n/d)$, $r \leqslant t_{\mathrm{non-DPLL}}$, $r_{\mathrm{bad}} = r/3$, and $r + r_{\mathrm{bad}} \leqslant r_{\max}(n, d, c)$, with extraneous set I_{bad}.

Given s such that $s/n = 2^{-o(d)}n$, let $\mathcal{D}$ be an (s, t)-myopic DPLL backtracking algorithm, where $t = r/3 = \Omega(n/d)$. Let $s' = \min\{s, \lfloor cr/6 \rfloor\} = 2^{-o(d)}n$. Then $\mathcal{D}$ is an (s', t)-myopic DPLL algorithm. By Lemma 7.1, there is a non-DPLL algorithm $\mathcal{S}(\mathcal{D})$ which produces the same result as $\mathcal{D}$ and takes at most 2^d as long. We will show that $\mathcal{S}(\mathcal{D})$ can be turned into a myopic algorithm too: that is, we will design an $(s, t_{\mathrm{non-DPLL}})$-myopic backtracking algorithm $\mathcal{S}'(\mathcal{D})$ which has the same backtracking tree as $\mathcal{S}(\mathcal{D})$. Theorem 2.10 tells us that Goldreich's function is secure against $(s, t_{\mathrm{non-DPLL}})$-myopic non-DPLL backtracking algorithms. Thus, Goldreich's function is secure against $\mathcal{S}'(\mathcal{D})$ and hence also against $\mathcal{D}$.

When finding a preimage $x \in f^{-1}(b)$ to Goldreich's function, the myopic DPLL algorithm $\mathcal{D}$, and therefore its simulation $\mathcal{S}(\mathcal{D})$, uses two sources of information. In order to behave in the same way as $\mathcal{S}(\mathcal{D})$, the myopic algorithm $\mathcal{S}'(\mathcal{D})$ must obtain both kinds of information. First, $\mathcal{S}(\mathcal{D})$ reads bits of b, but it reads no more than

$t = r/3$ bits before it has assigned values to s' variables. In order to have this information, $\mathcal{S}'(\mathcal{D})$ reads the same bits of b that $\mathcal{D}$ does. Second, $\mathcal{S}(\mathcal{D})$ knows at all times the set of all DPLL assignments that can be made. To obtain this knowledge, $\mathcal{S}'(\mathcal{D})$ will maintain a set $C \subseteq R$ of right-nodes of G, and $\mathcal{S}'(\mathcal{D})$ will make sure it has always read all the bits in C by reading b_i whenever it adds a node i to C. (The set C plays the same role here as in the proof of Lemma 6.7.) C will always contain every node in the extraneous set I_{bad}. In addition, $\mathcal{S}'(\mathcal{D})$ will maintain the following invariants:

– $C \setminus I_{\mathrm{bad}}$ is a closure for $\mathrm{Vars}(\rho) \setminus \Gamma(I_{\mathrm{bad}})$ in the graph $\hat{G} = G \setminus (I_{\mathrm{bad}} \cup \Gamma(I_{\mathrm{bad}}))$.
– $|C \setminus I_{\mathrm{bad}}| \leqslant 2c^{-1}|\mathrm{Vars}(\rho)|$.
– $|\partial C \setminus (\mathrm{Vars}(\rho) \cup \Gamma(I_{\mathrm{bad}}))| \leqslant c|C \setminus I_{\mathrm{bad}}|/2$.

By Lemma 6.6, applied to the graph $\hat{G}$ instead of the imperfectly expanding G, algorithm $\mathcal{S}'(\mathcal{D})$ can maintain these properties as long as it has assigned less than $s' \leqslant \lfloor cr/4 \rfloor$ variables. By Lemma 7.5, reading the bits in the set C is enough for $\mathcal{S}'(\mathcal{D})$ to know all the DPLL assignments that can be made, and so $\mathcal{S}'(\mathcal{D})$ has enough knowledge to behave in the same way as $\mathcal{S}(\mathcal{D})$. Before assigning values to s' variables, $\mathcal{S}'(\mathcal{D})$ reads at most $r/3$ bits of b_i because $\mathcal{D}$ read them, at most $r_{\mathrm{bad}} = r/3$ bits from I_{bad}, and at most $2c^{-1}s' = r/3$ bits from $C \setminus I_{\mathrm{bad}}$, so $\mathcal{S}'(\mathcal{D})$ is (s', r)-myopic, and therefore $(s', t_{\mathrm{non-DPLL}})$-myopic. □

8 MiniSat Experiment

Inverting Goldreich's function can be seen as the task of solving a constraint satisfaction problem with a planted solution. This suggests the use of a general-purpose SAT solver to solve the constraint satisfaction problem. We performed an experiment using MiniSat version 2.0 beta [11,10], which is one of the best publicly available SAT solvers. We always use the degree-five predicate $P_5(x) = x_1 \oplus x_2 \oplus x_3 \oplus (x_4 \wedge x_5)$. For each trial, we choose a new random graph of right-degree 5. MiniSat requires a boolean formula in conjunctive normal form as input, so we represent each constraint $P(x_{j_1}, x_{j_2}, x_{j_3}, x_{j_4}, x_{j_5}) = v_i$ by 16 clauses: one for each truth assignment to $x_{j_1}, \cdots, x_{j_5}$ that would violate the constraint.

We ran MiniSat on a Lenovo T61 laptop with 2GB of RAM and a 2.00GHz Intel T7300 Core Duo CPU. Fig. 1 plots the number of seconds taken to find a solution versus the input size n. The graph is plotted on a logarithmic scale. The time appears to grow exponentially in n.

9 Acknowledgements

The authors benefited from helpful discussions with Alekh Agarwal and Amir Shpilka.

References

1. Michael Alekhnovich, Edward A. Hirsch, and Dmitry Itsykson. Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas. *J. Autom. Reasoning*, 35:51–72, 2005.
2. Benny Applebaum. Pseudorandom generators with long stretch and low locality from random local one-way functions. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:7, 2011. Presented at the 44th ACM Symposium on Theory of Computing (STOC 2012).

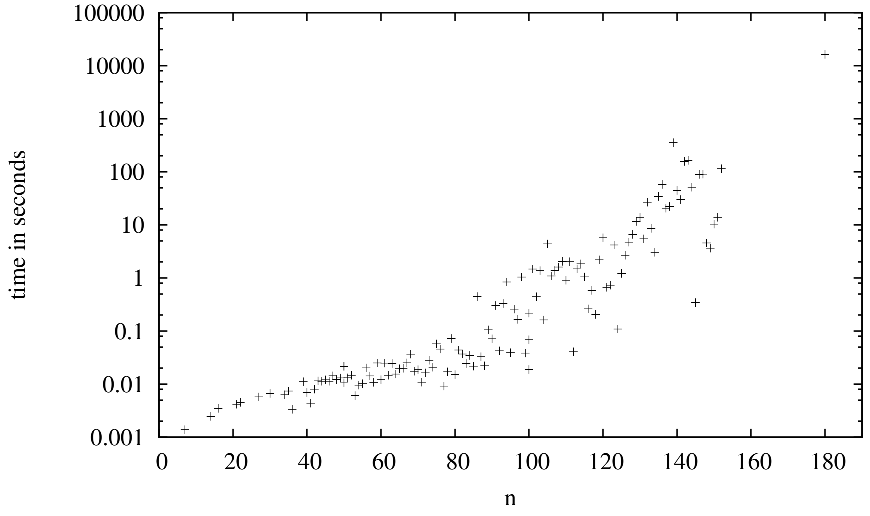

Fig. 1. Number of seconds taken by MiniSat to invert Goldreich's function for different values of n. We use the degree-five predicate $P_5(x) = x_1 \oplus x_2 \oplus x_3 \oplus (x_4 \wedge x_5)$ and a random bipartite graph of right-degree five.

3. Benny Applebaum, Boaz Barak, and Avi Wigderson. Public-key cryptography from different assumptions. In Leonard J. Schulman, editor, *STOC*, pages 171–180. ACM, 2010.
4. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC0. *SIAM J. on Computing*, 36(4):845–888, 2006.
5. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. On pseudorandom generators with linear stretch in NC0. In *APPROX-RANDOM*, pages 260–271, 2006.
6. Ben-Sasson and Wigderson. Short proofs are narrow–resolution made simple. *JACM: Journal of the ACM*, 48, 2001.
7. Andrej Bogdanov and Youming Qiao. On the security of goldreich's one-way function. In Irit Dinur, Klaus Jansen, Joseph Naor, and José D. P. Rolim, editors, *APPROX-RANDOM*, volume 5687 of *Lecture Notes in Computer Science*, pages 392–405. Springer, 2009.
8. James Cook, Omid Etesami, Rachel Miller, and Luca Trevisan. Goldreich's one-way function candidate and myopic backtracking algorithms. In Omer Reingold, editor, *Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings*, volume 5444 of *Lecture Notes in Computer Science*, pages 521–538. Springer, 2009.
9. Mary Cryan and Peter B. Miltersen. On pseudorandom generators in NC0. In *Proceedings of MFCS'01*, 2001.
10. Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In Fahiem Bacchus and Toby Walsh, editors, *SAT*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2005.
11. Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.
12. Oded Goldreich. Candidate one-way functions based on expander graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 7(90), 2000.
13. Dmitri Itsykson and Dmitry Sokolov. The complexity of inversion of explicit goldreich's function by DPLL algorithms. 2010.

14. Dmitry Itsykson. Lower bound on average-case complexity of inversion of goldreich's function by drunken backtracking algorithms. In Farid M. Ablayev and Ernst W. Mayr, editors, *CSR*, volume 6072 of *Lecture Notes in Computer Science*, pages 204–215. Springer, 2010.
15. Dmitry Itsykson and Dmitry Sokolov. Lower bounds for myopic DPLL algorithms with a cut heuristic. In Takao Asano, Shin-Ichi Nakano, Yoshio Okamoto, and Osamu Watanabe, editors, *ISAAC*, volume 7074 of *Lecture Notes in Computer Science*, pages 464–473. Springer, 2011.
16. Kazuo Iwama and Shuichi Miyazaki. Tree-like resolution is superpolynomially slower than dag-like resolution for the pigeonhole principle. In *Proceedings of ISAAC*, volume 1741 of *Lecture Notes in Computer Science*, pages 133–142, 1999.
17. Rachel Miller. Goldreich's one-way function candidate and drunken backtracking algorithms, 2009. Distinguished Majors Thesis.
18. Elchanan Mossel, Amir Shpilka, and Luca Trevisan. On ϵ-biased generators in NC^0. *Random Structures and Algorithms*, 29(1):56–81, 2006.
19. Saurabh Kumar Panjwani. An experimental evaluation of goldreich's one-way function. 2001.