Lower bound on average-case complexity of inversion of Goldreich's function by drunken backtracking algorithms *

Dmitry Itsykson
Steklov Institute of Mathematics at St. Petersburg
27 Fontanka, St.Petersburg, 191023, Russia.
dmitrits@pdmi.ras.ru.

March 13, 2012

Abstract

We prove an exponential lower bound on the average time of inverting Goldreich's function by *drunken* [AHI05] backtracking algorithms; therefore we resolve the open question stated in [CEMT09]. The Goldreich's function [Gol00] has n binary inputs and n binary outputs. Every output depends on d inputs and is computed from them by the fixed predicate of arity d. Our Goldreich's function is based on an expander graph and on the nonliniar predicates of a special type. Drunken algorithm is a backtracking algorithm that somehow chooses a variable for splitting and randomly chooses the value for the variable to be investigated at first. Our proof technique significantly simplifies the one used in [AHI05] and in [CEMT09].

1 Introduction

In 2000 Goldreich introduced the candidate one-way function based on expanders [Gol00]. The function has n binary inputs and n binary outputs. Every output depends on d inputs and is computed from them by the fixed predicate of arity d. Goldreich suggested using expander graphs as graphs of dependency and a random predicate of arity d. There are many similarities between the Goldreich's function and the pseudorandom generator by Nisan and Wigderson [NW94].

One of the approaches for inverting of one-way function is the usage of contemporary SAT solvers [MZ06]. Almost all SAT algorithms are based on backtracking (so called DPLL (by names of authors Davis, Putnam, Logeman, Loveland) algorithms [DP60, DLL62]). Backtracking algorithm is a recursive algorithm. On each recursive call it

*Partially supported by RFBR grants 08-01-00640 and 09-01-12137-ofi_m, the Fundamental research program of the Russian Academy of Sciences, the president grants NSh-5282.2010.1 and MK-4089.2010.1 and by Federal Target Programme "Scientific and scientific-pedagogical personnel of the innovative Russia" 2009-2013.

simplifies an input formula F (without affecting its satisfiability), chooses a variable v and makes two recursive calls on the formulas $F[v := 1]$ and $F[v := 0]$ in some order. It returns the result "Satisfiable" if at least one of recursive calls returns "Satisfiable" (note that it is not necessary to make the second call if the first one was successful). Recursion stops if the input formula becomes trivial. That is, the algorithm is only allowed to backtrack when unsatisfiability in the current branch is proved. A backtracking algorithm is defined by simplification rules and two heuristics: the heuristic **A** chooses a variable for splitting and the heuristic **B** chooses a value that will be investigated first.

Lower bounds on the running time of backtracking algorithms on unsatisfiable formulas follow from lower bounds on the size of resolution proofs [Tse68]. Unsatisfiable formulas based on a pseudorandom generator by Nisan and Wigderson are used for proving lower bounds in several propositional proof systems [ABSRW00]. Note that formulas that code the problem of inverting a one-way function are usually satisfiable. If we do not restrict a type of heuristics of backtracking algorithms, then the exponential lower bound on running time of backtracking algorithms implies $\mathbf{P} \neq \mathbf{NP}$ (otherwise heuristic **B** may compute the correct value of the variable in polynomial time).

The first unconditional lower bounds on running time of backtracking algorithms on satisfiable formulas were proved in [AHI05] for *myopic* and *drunken* algorithms. Heuristics in myopic algorithms are restricted in the following way: they are allowed to read only a small fraction of the formula precisely, but they can see the rest of the formula sketchy (for example in [AHI05] they don't see negations but have access to the statistics of number of positive and negative variable occurrences). Exponential lower bound on the running time of myopic algorithms was proved on the family of formulas that actually code the problem of inverting Goldreich's function based on linear predicate. In drunken algorithms heuristic **A** has no restriction (and actually it may be not computable), but heuristic **B** selects the value just at random. For drunken algorithms hard satisfiable formulas are based on any family of hard unsatisfiable formulas.

Goldreich's function based on a linear predicate is not very interesting since it may be inverted by Gaussian elimination. In the paper [CEMT09] the technique from [AHI05] was extended for proving lower bounds for myopic algorithms on nonlinear predicates. In particular it was proved in [CEMT09] that any myopic algorithm has exponential running time in average case when it solves a problem of inverting Goldreich's function based on the predicate $x_1 \oplus x_2 \oplus \cdots \oplus x_{d-2} \oplus x_{d-1}x_d$. The paper [CEMT09] also presents results of experimental analysis of running time of contemporary SAT solvers on the problem of inverting Goldreich's function with the above predicate. Their analysis shows that these formulas are hard for MiniSAT 2.0 [EB05, ES03]. The question of exponential lower bound on inverting Goldreich's function by a drunken algorithm was left open in [CEMT09]. In this paper we give an answer on this question. In particular we prove that the average running time of drunken algorithms on formulas that code the problem of inverting Goldreich's function based on a random graph and the predicate $x_1 \oplus \cdots \oplus x_{d-k} \oplus Q(x_{d-k+1}, \ldots, x_d)$ is exponential with high probability. Here Q is an arbitrary predicate of arity k, $k + 1 < \frac{d}{4}$ and d is a constant large enough.

The proof strategy is as follows: at first we prove a lower bound for unsatisfiable formulas using the technique from [BSW01], then we show that it is possible to introduce some superstructure on drunken algorithms, which does not increase the running time but guarantees that in the first several steps the algorithm does not backtrack. After that

we show that with high probability Goldreich's function has a small number of pre-images and with high probability the algorithm with superstructure falls into an unsatisfiable formula and we will apply a lower bound for unsatisfiable formulas. The general plan of our proof is the same as it was in [AHI05] and [CEMT09], but the resulting proof is substantially simpler.

We also show that drunken algorithms are powerful enough: they may solve satisfiable Tseitin formulas in polynomial time while unsatisfiable Tseitin formulas are hard for all backtracking algorithms. Drunken algorithms may also simulate the pure literal simplification rule while myopic algorithms from [CEMT09] are not allowed to use this rule.

2 Preliminaries

Propositional variable is one that has 0/1-value, literal is either a variable or its negation. A clause is a disjunction of literals, a CNF formula is a conjunction of clauses. A k-CNF formula is a CNF formula in which all clauses contain at most k literals. The formula is satisfiable if there exists substitution for its variables such that the value of the formula becomes 1 (we call such substitution a satisfying assignment).

The set of all functions from $\{0,1\}^n$ to $\{0,1\}^n$ we denote as $\mathfrak{F}_n$. For every function $f \in \mathfrak{F}_n$ and every string $b \in \{0,1\}^n$ the equality $f(x) = b$ may be written as a CNF formula with propositional variables $x_1, \ldots, x_n$. We denote such formula $\Phi_{f(x)=b}$.

In this paper $G(V,E)$ is a bipartite graph with multi-edges. The vertices of G are divided into two parts: $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$; the number of vertices in each part is n. Vertices in X are inputs and vertices in Y are outputs. Every vertex in Y has degree d.

Proposition 2.1 (Chernoff-Hoefding bounds)**.** Independent and equally distributed random variables $X_1, X_2, \ldots, X_N$, such that for every $1 \le i \le N$ $X_i \in \{0,1\}$ and $E[X_i] = \mu$, satisfy the following inequality: $\Pr\{|\frac{\sum_{i=1}^{N} X_i}{N} - \mu| \ge \epsilon\} \le 2e^{-2\epsilon^2 N}$.

Goldreich introduced the function $g : \{0,1\}^n \to \{0,1\}^n$ based on a bipartite graph $G(V,E)$ and a predicate $P : \{0,1\}^d \to \{0,1\}$ [Gol00]. Every string from $\{0,1\}^n$ defines a value of inputs $\{x_1, x_2, \ldots, x_n\}$; the value $(g(x))_j$ (j-th symbol of $g(x)$) may be computed as follows: if y_j is adjacent with $x_{j_1}, x_{j_2}, \ldots, x_{j_d}$, then $(g(x))_j = P(x_{j_1}, x_{j_2}, \ldots, x_{j_d})$. We assume that every vertex in Y has some order on the incoming edges. Goldreich suggested using random predicates and expanders.

The problem of inverting of function g on the string b (i.e. equation $g(x) = b$) may be written as a d-CNF formula $\Phi_{g(x)=b}$: every equality $P(x_{j_1}, x_{j_2}, \ldots, x_{j_d}) = b_j$ we write as a d-CNF formula of at most 2^d clauses. For every set of vertices $A \subseteq Y$ the formula $\Phi^A_{g(x)=b}$ denotes the subformula of $\Phi_{g(x)=b}$ that consists of all clauses corresponding to vertices in A.

Let G be a bipartite graph, its vertices are split into two parts $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$. For $A \subseteq Y$ we denote the set of all vertices in X that are connected with at least one vertex in A as $\Gamma(A)$ (neighbours of A); and denote the set of all vertices in X that are connected with exactly one vertex in A by one edge as $\delta(A)$ (boundary of A).

Definition 2.1. The graph G is a (r, d, c)-expander, if 1) the degree of any vertex in Y is equal to d; 2) for any set $A \subseteq Y, |A| \leq r$ we have $\Gamma(A) \geq c|A|$. The graph G is called a (r, d, c)-boundary expander if the second condition is replaced by: 2) for any set $A \subseteq Y, |A| \leq r$ we have $\delta(A) \geq c|A|$.

Lemma 2.1 ([AHI05], lemma 1)**.** Every (r, d, c)-expander is also a $(r, d, 2c-d)$-boundary expander.

Proof. Let $A \subseteq Y$, $|A| \leq r$, then $|\Gamma(A)| \geq c|A|$. The number of edges between A and $\Gamma(A)$ may be estimated: $d|A| \geq |\delta(A)| + 2|\Gamma(A) \setminus \delta(A)| = 2|\Gamma(A)| - |\delta(A)| \geq 2c|A| - \delta(A)$. Finally we get $\delta(A) \geq (2c - d)|A|$. □

Lemma 2.2 ([HLW06], lemma 1.9)**.** For $d \geq 32$, for all big enough n a random bipartite d-regular graph, where parts X and Y contain n vertices is a $(\frac{n}{10d}, d, \frac{5}{8}d)$-expander with probability 0.9 if for every vertex in Y, d edges are chosen independently at random (with repetitions).

Corollary 2.1. In terms of Lemma 2.2 this graph is a $(\frac{n}{10d}, d, \frac{1}{4}d)$-boundary expander.

Proof. Follows from Lemma 2.1. □

Let $f \in \mathfrak{F}_n$ be some function. The problem of finding a satisfying assignment of $\Phi_{f(x)=b}$ and the problem of finding an element in $f^{-1}(b)$ are equivalent.

We consider a wide class of SAT algorithms: backtracking algorithms. A backtracking algorithm is defined by two *heuristics* (procedures): 1) Procedure **A** maps a CNF formula to one of its variables. (This is a variable for splitting). 2) Procedure **B** maps CNF formula and its variable to $\{0, 1\}$. (This value will be investigated at first).

An algorithm may also use some syntactic *simplification rules*. Simplification rules may modify the formula without affecting its satisfiability and also make substitutions to its variables if their values can be inferred from a satisfiability of the initial formula.

The backtracking algorithm is a recursive algorithm. Its input is a formula φ and a partial substitution ρ.

Algorithm 2.1. Input: formula φ and substitution ρ

- Simplify φ by means of simplification rules (assume that simplification rules change φ and ρ; all variables that are substituted by ρ should be deleted from φ).
- If current formula is empty (that is, all its clauses are satisfied by ρ), then return ρ. If formula contains an empty clause (unsatisfiable), then return "formula is unsatisfiable".
- $x_j := \mathbf{A}(\varphi)$; $c := \mathbf{B}(\varphi, x_j)$
- Make a recursive call with the input $(\varphi[x_j := c], \rho \cup \{x_j := c\})$, if the result is "formula is unsatisfiable", then make a recursive call with the input $(\varphi[x_j := 1 - c], \rho \cup \{x_j := 1 - c\})$ and return its result, otherwise return the result of the first recursive call.

Definition 2.2. Drunken algorithms [AHI05] are backtracking algorithms, where the heuristic **A** may be arbitrary (even not computable) and the heuristic **B** chooses the value of variable at random with equal probabilities. Simplification rules: 1) *Unit clause elimination*: if formula contains a clause with only one literal, then make a substitution that satisfies that clause. 2) *Pure literals rule*: if formula contains a variable that has only positive or only negative occurrences, then substitute it with the corresponding value.

In Section 3 we will show that we may assume that a drunken algorithm does not use the above simplification rules. Usages of them may be replaced by an appropriate choice of splitting variable.

Running time of a backtracking algorithm for a given sequence of random bits is the number of recursive calls.

3 What can we solve by drunken algorithms?

In this section we show that it is possible to modify a drunken algorithm in such a way that it will not use pure literals and unit clause elimination rules while its running time is increased only polynomially. We also show that there exists a drunken algorithm that solves satisfiable Tseitin formulas in polynomial time.

Proposition 3.1. For any drunken algorithm $\mathcal{A}$ there exists another drunken algorithm $\mathcal{B}$, that does not use unit clause elimination rule. The running time of algorithm $\mathcal{B}$ (for a given sequence of random bits) is at most the running time of $\mathcal{A}$ (for the same sequence of random bits) times n, where n is the number of variables in the input formula.

Proof. If the current formula contains a unit clause, then the algorithm $\mathcal{B}$ makes splitting on the variable of this unit clause. One of the branches will result in a trivially unsatisfiable formula, in the other branch resulting formula will be the same as after application of unit clause elimination rule. □

Proposition 3.2. For any drunken algorithm $\mathcal{A}$ there exists another drunken algorithm $\mathcal{C}$ that does not use unit clause elimination and pure literals rules. The running time of algorithm $\mathcal{C}$ is at most the running time of $\mathcal{A}$ times n^2, where n is the number of variables in the input formula.

Proof. Note that an application of the pure literals rule is just a removing of some clauses. If formula is unsatisfiable, it is possible not to use pure literals rule (a heuristic **A** may remove corresponding clauses by itself). There is a problem for satisfiable formulas: it is possible that in some moment the current formula is completely simplified by an application of a sequence of pure literals rule. Let's consider satisfiable formula that may be completely simplified (up to the formula with an empty set of clauses) by application of a sequence of pure literals rules. Let the sequence of pure literals be $l_1, l_2, \ldots, l_t$. Let the application of the pure literals rule (in the above order) to the literal l_i eliminates the set of clauses C_i. Let m be the number of variables in the formula. By induction on $t+m$ we show that it is possible not to use pure literals rule. The base of induction $t + m = 0$ is trivial. Induction step. Consider the nonempty set of clauses C_t; let X be the set of variables that have occurrences in clauses of C_t and are not equal to the variable of literal

l_t. We consider two cases. In the first case $X = \emptyset$, then all clauses in C_t are unit clauses with literal l_t, then we may apply unit clause elimination rules, so the number of pure literals rules will be reduced to $t-1$ and we apply induction hypothesis. In the second case $X \neq \emptyset$, then we make a substitution to one variable in the set X (note that the variables in X are not equal to any of variables of $l_1, l_2, \ldots, l_t$). After this substitution the formula still can be simplified by an application of t pure literals rules but the number of variables is decreased by one and we use induction hypothesis. Finally we remove all pure literals rules and make at most n substitution in every formula, where the satisfying assignment was obtained by multiple applications of pure literals rules. We also remove unit clause elimination rules by application of Proposition 3.1. □

Further we assume that drunken algorithms don't use simplifications rules.

Now we show that drunken algorithms may efficiently solve Tseitin formulas. Tseitin formula is based on a simple connected undirected graph $H(V, E)$ with degree bounded by a constant d. Every edge $e \in E$ has the corresponding propositional variable p_e. There is a function $f : V \to \{0, 1\}$; for every vertex $v \in V$ we put down a formula in CNF that codes an equality $\bigoplus_{u \in V : (u,v) \in E} p_{(u,v)} = f(v)$. ($\oplus$ denotes the summation modulo 2). The conjunction of formulas described above is called Tseitin formula. If $\bigoplus_{v \in V} f(v) = 1$, then Tseitin formula is unsatisfiable. Indeed, if we sum (modulo 2) all equalities stated in vertices we get $0 = 1$ since every variable has exactly 2 occurrences. If $\bigoplus_{v \in V} f(v) = 0$, then Tseitin formula is satisfiable ([Urq87], Lemma 4.1).

Satisfiable Tseitin formulas may be solved in polynomial time by appropriate drunken algorithm as follows. We assume that the substitution of a variable removes the corresponding edge from the graph. While the graph contains cycles, the drunken algorithm chooses an edge from a cycle for the substitution. Note that after each substitution the current formula remains satisfiable Tseitin formula (if we substitute 0, the function f for the substituted formula is the same as for the original one and if we substitute 1, the function f for substituted formula differs from the original one in 2 ends of the edge). If the graph becomes a tree, then it contains a vertex of degree 1, then the formula contains a unit clause and the whole formula can be solved by application of unit clause elimination rules (we may remove them by Proposition 3.1).

4 Behavior of drunken algorithms on unsatisfiable formulas

Behavior of backtracking algorithms on unsatisfiable formulas is closely connected with the resolution proof system. The resolution proof system is used for proving of unsatisfiability of CNF formulas. The proof of unsatisfiability of formula φ in the resolution proof system is a sequence of clauses, every clause in this sequence is either a clause of φ or a result of application of the resolution rule to two previous clauses; and the last clause in the sequence is an empty clause (a contradiction). The resolution of two clauses $(l_1 \vee l_2 \vee \cdots \vee l_n)$ and $(l'_1 \vee l'_2 \vee \cdots \vee l'_m)$ where $l'_m = \neg l_n$ is the clause $(l_1 \vee \cdots \vee l_{n-1} \vee l'_1 \vee \cdots \vee l'_{m-1})$. The proof is called treelike if every inferred clause is used as the premise of the resolution rule at most once.

The running of every drunken algorithm on the unsatisfiable formula corresponds to the splitting tree. Vertices of the tree are marked with variables that are chosen for splitting. There are two outgoing edges from every vertex except leaves; one of the edges is marked with 0, the other edge is marked with 1. In every leaf at least one of clauses of initial formula is refuted. The running time of a drunken algorithm is the size of the splitting tree (note that if formula is unsatisfiable then the algorithm should investigate the whole tree and it's number of steps is the same for all random choices).

The following statement is well known.

Proposition 4.1. The running time of a drunken algorithm on unsatisfiable formula is at least the size (number of clauses) of the shortest treelike resolution proof.

Proof. By induction of the size of the tree it is easy to show that if unsatisfiable formula φ has splitting tree of size k then φ has resolution refutation of size k. The base of induction is the spitting tree with only one vertex, such formula should contain an empty clause therefore the size of resolution refutation is 1. Induction step. Note that the tree necessarily contains two leaves u and v with the same parent w. Let x_i be the splitting variable in the vertex w, the leaf u corresponds to the assignment $x_i = 1$ and the leaf v corresponds to the assignment $x_i = 0$. Two clauses that are refuted in the vertices v and u contain the variable x_i with different signs. The resolvent (the result of an application of the resolution rule) of this two clauses C must be refuted in the vertex w. We construct new splitting tree: cut leaves u and w and add clause C to vertex w. Now we get a correct splitting tree for a formula that is obtained from the initial formula by adding a resolvent of two clauses. And we apply induction hypothesis to the resulting tree (the number of vertices is decreased by 1).

□

Ben-Sasson and Wigderson in [BSW01] introduced the notion of width of the proof. The width of a clause is the number of literals in it. The width of a CNF formula is the width of its widest clause. The width of a resolution proof is the width of its widest clause.

Theorem 4.1 ([BSW01], corollary 3.4)**.** The size of a treelike resolution refutation of the formula φ is at least 2^{w-w_φ}, where w is the minimum width of the resolution refutation of φ and w_φ is the width of φ.

Let G be a boundary (r, d, c)-expander. We associate a proposition variable with every vertex in set X. Let every vertex y_j in set Y have a CNF formula that depends on variables adjacent to y_j. We denote the formula in the vertex y_j as φ_j. Obviously the width of φ_j is at most d. The conjunction of all formulas that correspond to the vertices Y we denote Φ. For any subset $A \subseteq Y$ the conjunction of all formulas that correspond to the vertices in A we denote as Φ^A.

We say that a variable is *sensible* if by changing its value we change the value of the formula (for every assignment of values of other variables).

Theorem 4.2. Let every formula φ_j contain at most k insensible variables; ρ is a partial assignment to variables of X such that formula $\Phi|_\rho$ is unsatisfiable and for any set of vertices $A \subseteq Y$, $|A| < \frac{r}{2}$, the formula $\Phi|_\rho^A$ is satisfiable. Then any resolution proof of $\Phi|_\rho$ has width at least $\frac{(c-k)r}{4} - |\rho|$.

Proof. We consider Ben-Sason-Wigderson measure μ that is defined on the clauses of resolution proof of $\Phi|_\rho$. $\mu(D)$ is the size of the minimal set of vertices A such that clause D is a semantic implication of $\Phi^A|_\rho$ (it means that every satisfying assignment of $\Phi^A|_\rho$ also satisfies D). The measure μ is semiadditive: if clause D is a resolvent of clauses C_1 and C_2, then $\mu(D) \leq \mu(C_1) + \mu(C_2)$. Since for every set $A \subseteq Y$ such that $|A| < \frac{r}{2}$, formula $\Phi|_\rho^A$ is satisfiable, then the measure of an empty clause is at least $\frac{r}{2}$. Semiadditivity implies that there exists a clause C such that $\frac{r}{2} > \mu(C) \geq \frac{r}{4}$ for r large enough. Let A be the minimal set of vertices such that $\Phi|_\rho^A$ semantically implies C, i.e. $|A| = \mu(C) \geq \frac{r}{4}$. Since G is a (r, d, c)-boundary expander we have $\delta(A) \geq c|A|$. $\delta(A)$ is a set of variables that have exactly one occurrence in the formulas corresponding to the set A. There are at least $(c-k)|A|$ variables among them that are sensible for at least one vertex of A. There are at least $(c-k)|A| - |\rho|$ sensible variables in the formula $\Phi|_\rho^A$. Now we will show that the clause C contains all sensible variables. Suppose for contradiction that there is a variable x_j that is sensible for a vertex $v \in A$ and the clause C doesn't contain x_j. Consider the set $A \setminus \{v\}$. It doesn't semantically imply C, therefore there exists such an assignment that satisfies all formulas for $A \setminus \{v\}$ and doesn't satisfy C. We may change the value of x_j in this assignment in such way that the resulting assignment satisfies all formulas in A and doesn't satisfy C. The later contradicts the fact that C is a semantic implication of A. □

Corollary 4.1. The size of the splitting tree of $\Phi|_\rho$ is at least $2^{\frac{(c-k)r}{4} - |\rho| - d}$.

Proof. Follows from the Theorem 4.2, Theorem 4.1 and Proposition 4.1. □

5 Behaviour of drunken algorithms on satisfiable formulas

Let G be a bipartite boundary (r, d, c)-expander. Let $c > k + 1$.

Definition 5.1. Let $J \subseteq X$, the set of vertices $I \subseteq Y$ is called k-closure of the set J if there is a finite sequence of sets $I_1, I_2, \ldots, I_m$ (we denote $C_\ell = \bigcup_{1 \leq i \leq \ell} I_i$, $C_0 = \emptyset$), such that the following properties are satisfied:

- $I_\ell \subseteq Y$ and $0 < |I_\ell| \leq \frac{r}{2}$ for all $1 \leq \ell \leq m$;
- $I_i \cap I_j = \emptyset$ for all $1 \leq i, j \leq m$;
- $|\delta(I_\ell) \setminus (\Gamma(C_{\ell-1}) \cup J)| \leq (1+k)|I_l|$; for all $1 \leq \ell \leq m$;
- for all $I' \subseteq Y \setminus C_m$ if $0 < |I'| \leq \frac{r}{2}$, then $|\delta(I') \setminus (\Gamma(C_m) \cup J)| > (1+k)|I'|$;
- $I = C_m$.

The set of all k-closures of the set J we denote as $Cl^k(J)$.

Lemma 5.1. 1. For every set $J \subseteq X$ there exists a k-closure. 2. Let $J_1 \subseteq J_2$, then for every $I_1 \in Cl^k(J_1)$ there exists $I_2 \in Cl^k(J_2)$ such that $I_1 \subseteq I_2$

Proof. 1. A k-closure may be obtained as a result of the following algorithm $\mathcal{C}$ on the input $(J, \emptyset)$.

Algorithm 5.1. Algorithm $\mathcal{C}(J, I_0)$

1. $I := I_0$ (the variable I means some subset of Y)
2. While there exists $I' \subseteq Y \setminus I$ such that $0 < |I'| \leq \frac{r}{2}$, $|\delta(I') \setminus (\Gamma(I) \cup J)| \leq (1+k)|I'|$
   - $I := I \cup I'$
3. Return I.

2. Let $I_1 \in Cl^k(J_1)$, then we can get $I_2 \in Cl^k(J_2)$ as a result of the algorithm $\mathcal{C}$ on the input (J_2, I_1). The condition $I_1 \subseteq I_2$ is satisfied. □

Lemma 5.2 ([AHI05])**.** Let $|J| < \frac{(c-k-1)r}{2}$, then for every set $I \in Cl^k(J)$ the inequality $|I| \leq (c-k-1)^{-1}|J|$ is satisfied

Proof. Proof by contradiction. Let $I_1, I_2, \ldots, I_m$ be the sequence corresponding to the k-closure I, $C_\ell = \bigcup_{1 \leq i \leq \ell} I_\ell$. Let t be the minimal number such that the inequality $|C_t| > (c-k-1)^{-1}|J|$ is satisfied, then $|C_t| \leq (c-k-1)^{-1}|J| + \frac{r}{2} \leq r$. Then $|\delta(C_t)| \geq c|C_t| > |J| + (k+1)|C_t|$. By induction on ℓ we will show that $|\delta(C_\ell) \setminus J| \leq (k+1)(|C_\ell|)$, and it contradicts the above inequality for $l = t$. If $l = 1$ the inequality follows from $|\delta I_1 \setminus J| \leq (k+1)|I_1|$. $|\delta(C_\ell) \setminus J| \leq |\delta(I_1 \cup \cdots \cup I_{\ell-1}) \setminus J| + |\delta(I_\ell) \setminus (J \cup \Gamma(C_{\ell-1}))| \leq (k+1)((|C_{\ell-1}|)) + (k+1)|I_\ell|$. □

We assume that a drunken algorithm creates a splitting tree during the execution. At the beginning it creates the root of the tree that becomes the current vertex. Each vertex of the tree has a current formula, each edge is associated with the assignment of one variable. The path from the root to the vertex defines a partial assignment that is a union of all assignments along this path. The current vertex of the tree becomes a leaf if the current formula is either already satisfied (i.e. all its clauses are satisfied) or contains an empty clause (i.e. a contradiction). In the first case the algorithm prints a satisfying assignment and stops. In the second case the algorithm looks for the closest backtrack point along the path to the root and considers the vertex with that backtrack point as current (in this case we say that the algorithm backtracks). If there are no vertices with a backtrack point, then the algorithm stops and returns "formula is unsatisfiable". If the current formula is not trivially unsatisfiable or satisfiable, then the algorithm chooses the variable for splitting and the value for splitting according to heuristics **A** and **B**, puts a backtrack point in the vertex and creates a descendant that corresponds to the assignment that was chosen; this descendant becomes the current vertex. If the current vertex has a backtrack point, then the algorithm removes this point and creates a descendant corresponding to the assignment that was not investigated in that vertex.

Now we describe a superstructure of drunken algorithms that slightly modifies their behavior on the formula $\Phi_{g(x)=b}$ for several first steps. After this the superstructure finishes its work and the algorithm continues its normal work without modification. We claim that the running time of the algorithm with the superstructure is not increased. (The last statement is not very clear since our algorithm uses random bits. In our case

it should be understood in the following way: the original algorithm uses p random bits and the algorithm with the superstructure uses q bits where $q \leq p$, and for every string of random bits r of length q there are 2^{p-q} strings of random bits of length p such that the running time of the original algorithm on those strings is at least the running time of the algorithm with the superstructure on the string r. The above correspondence covers all strings of length p.)

The superstructure has a partial assignment π. If π assigns some value to the variable x, then we call x forced. If a drunken algorithm tries to make an assignment to a forced variable that differs from the value in π, then the superstructure doesn't allow this. In other words, the superstructure cuts off a subtree but we guarantee that the cut subtree is unsatisfiable (it does not contain a satisfying assignment). We also guarantee that while the superstructure is working there are no backtrackings (all backtrackings are left in the cut subtrees).

Let's formally describe the superstructure. Let drunken algorithm $\mathcal{A}$ get as input a satisfiable formula $\Phi_{g(x)=b}$, where G is a (r, d, c)-boundary expander, k is the number of insensible variables of the predicate P and $k + 1 < c$.

1. $J := \emptyset$, $I := \emptyset$, $\rho := \emptyset$ (current substitution)
2. $\pi := \emptyset$ (initially there are no forced variables).
3. While $|J| < \frac{r(c-k-1)}{16d}$ and $|\rho| < n$ do
   (a) If algorithm $\mathcal{A}$ is ready to finish its work or it wants to backtrack, then break.
   (b) Let $\mathcal{A}$ choose a variable x_j for the splitting.
   (c) If variable x_j is forced and π contains assignment $x_j := a$, then $\rho := \rho \cup \{x_j := a\}$. In the splitting tree we add one decender and we do not put a backtracking point.
   (d) Otherwise the variable x_j is not forced, then
   - Let $\mathcal{A}$ chooses value a, then $J := J \cup \{x_j\}$, $\rho := \rho \cup \{x_j := a\}$. We put backtrack point in the current vertex.
   - We extend I to the element of $Cl^k(J)$ (it is possible by the item (2) of Lemma 5.1).
   - For all variables x_j from $\Gamma(I)$ and $a \in \{0, 1\}$, if the value $x_j = a$ is a semantic implication of formula $\Phi^I_{g(x)=b}|_\rho$, then $\pi := \pi \cup \{x_j := a\}$. (Formally it is possible that the formula $\Phi^I_{g(x)=b}|_\rho$ implies both $x_j = 0$ and $x_j = 1$. In this case we add to π only one of them; later we show that this case is impossible).
   - Create a descendant in the tree that corresponds to the made assignment, this descendant becomes the current vertex.
4. Simulate $\mathcal{A}$ without changes on the formula $\Phi_{g(x)=b}|_\rho$ in the current vertex of the tree.

Let the loop at the 3rd step of the superstructure be executed (up to the end) t times. For $0 \leq i \leq t$ we denote as J_i, I_i, ρ_i the values of the variables J, I, ρ before the $(i+1)$-th

iteration of the loop at the 3rd step. (I_t, J_t, ρ_t are the values after t-th iteration of the loop).

The following Lemma implies that during the work of the superstructure the algorithm does not backtrack.

Lemma 5.3. For every $0 \leq i \leq t$ and for any subset $A \subseteq Y$, such that $|A| \leq \frac{r}{2}$, the formula $\Phi^A_{g(x)=b}|_{\rho_i}$ is satisfiable and $I_i = Cl^k(J_i)$.

Proof. Proof by induction on i. Let's verify the condition for $i = 0$, $\rho_0 = \emptyset$. Proof by contradiction. We consider the smallest set $A \subset Y$, $|A| \leq \frac{r}{2}$ such that the formula $\Phi^A_{g(x)=b}$ is unsatisfiable. Since G is a boundary expander, then $|\delta(A)| \geq c|A|$, and therefore there are at least $(c-k)|A|$ sensible boundary variables for formulas from the set A. By changing of the value of the boundary variable we may change the value of some formula from A, that is this formula (and a vertex) may be removed from A; the latest contradicts to the minimality of A.

The induction step. Proof by contradiction. Consider the minimum set $A \subseteq Y$, $|A| \leq \frac{r}{2}$ such that the formula $\Phi^A_{g(x)=b}|_{\rho_{i+1}}$ is unsatisfiable. Let $A_1 = A \cap I_{i+1}$, $A_2 = A \setminus I_{i+1}$. If A_2 is not empty, then the definition 5.1 implies $|\delta(A_2) \setminus (\Gamma(I_{i+1}) \cup J)| > (k+1)|A_2|$, therefore we may remove at least one vertex from A_2 since one of the formulas in A_2 may be satisfied by a sensible boundary variable (a sensible boundary variable exists since each vertex has at most k insensible variables) that contradicts to the minimality of A. Therefore $A_2 = \emptyset$ and $A \subseteq I_{i+1}$.

We split A on $A' = A \cap I_i$ and $A'' = A \setminus I_i$. Let $A'' = \emptyset$. By the induction hypothesis the formula $\Phi^{A'}_{g(x)=b}|_{\rho_i}$ is satisfiable since $|A'| \leq |A| \leq \frac{r}{2}$. The formula $\Phi^{A'}_{g(x)=b}|_{\rho_{i+1}}$ is satisfiable since ρ_{i+1} differs from ρ_i in only one assignment and the variables are forced by the substitution π only if their values are semantic implications of the formula $\Phi^{I_i}_{g(x)=b}|_{\rho_i}$. The latest means that it is impossible for the algorithm $\mathcal{A}$ (on the $(i+1)$-the iteration of the loop) to make $\Phi^{I_i}_{g(x)=b}|_{\rho_i}$ unsatisfiable by one assignment.

Let $A'' \neq \emptyset$. $|A''| \leq \frac{r}{2}$ and $A'' \cap I_i = \emptyset$ imply $|\delta(A'') \setminus (\Gamma(I_i) \cup J)| > (k+1)|A''|$, that is the set A'' contains at least two sensible boundary variables (that are not in $\Gamma(I_i) \cup J$), therefore after one assignment there is at least one sensible boundary variable. We can remove at least one vertex from A'' with $\Phi^A_{g(x)=b}|_{\rho_{i+1}}$ remaining unsatisfiable. This contradicts the minimality of A. □

Now we show that the algorithm can't find the satisfying assignment during the work of the superstructure. During the work of the superstructure for every $0 \leq i \leq t$ the inequality $|J_i| \leq \frac{r(c-k-1)}{16d}$ is satisfied. Lemma 5.2 implies $|I_i| \leq \frac{r}{16d}$, hence $|\Gamma(I_i)| \leq \frac{r}{16}$. The number of variables that were assigned during the work of the superstructure is at most $|\Gamma(I_t)| \cup |J_t| \leq \frac{r}{8}$ ($|J_t| \leq \frac{r}{16}$ since $c \leq d$ in the (r,d,c)-boundary expander). This is not enough to satisfy the formula, since any subset $A \subseteq Y$ of size r contains at least r sensible variables. To satisfy the formula we should assign a value to all sensible variables.

Lemma 5.4. Let g be the Goldreich's function based on the (r,d,c)-boundary expander G and the predicate P, which has at most k insensible variables and $c > k+1$. Let b be the n-th bit string such that the equation $g(x) = b$ has at most $2^{\frac{r(c-k-1)}{64d}}$ solutions. Then with probability $1 - 2^{-\Omega(r)}$ the running time of a drunken algorithm on the formula $\Phi_{g(x)=b}$ is $2^{\Omega(r)}$ (in asymptotic notations c, k and d are considered to be constants).

Proof. Since the superstructure doesn't increase the running time it is sufficient to estimate the running time of the algorithm with the superstructure. Since the superstructure works until $|J| \leq \frac{r(c-k-1)}{16d}$ and $|I| \in Cl^k(J)$ we have that Lemma 5.2 implies $|I| \leq \frac{r}{16d}$. Therefore $|\Gamma(I)| \leq \frac{r}{16}$. Hence during the work of the superstructure the number of assignments made is at most $|\Gamma(I)| + |J| \leq \frac{r}{8}$. The set J corresponds to splittings (to assignments that put a backtrack point). The first substituted values are chosen at random. The substitution of one variable $x_j := a$ is *lucky* for a formula φ if it agrees with at least half of satisfying assignments of φ and *unlucky* otherwise.

During the work of the superstructure a drunken algorithm makes $\frac{r(c-k-1)}{16d}$ assignments choosing the values at random. With probability $\frac{1}{2}$ the chosen value is unlucky, that is, the number of satisfying assignments decreases at least by half. Chernoff bound implies that with probability $1 - 2^{-\Omega(r)}$ there are at least $\frac{r(c-k-1)}{64d}$ unlucky assignments. Thus with probability $1 - 2^{-\Omega(r)}$ after the work of the superstructure the current formula is unsatisfiable; the size of substitution ρ is at most $\frac{r}{8}$. The statement of the Lemma follows from Corollary 4.1, where we've proved the lower bound for unsatisfiable formulas. □

Theorem 5.1 (cf. [CEMT09], theorem 4.1)**.** Let $P_d(x_1, x_2, \ldots, x_d) = x_1 \oplus x_2 \oplus \cdots \oplus x_{d-k} \oplus Q(x_{d-k+1}, \ldots, x_d)$, where Q is an arbitrary predicate of arity k and $k + 1 < \frac{d}{4}$. The graph G is obtained randomly in the following way: for every vertex in Y we choose independently at random d edges to X (repetitions are allowed). Then $\mathrm{E}[\#(x, y) \mid g(x) = g(y)] = 2^{(1+2^{-\Omega(d)})n}$, where g is a Goldreich's function based on G and the predicate P_d.

Proof. See appendix A. □

Now we prove the main theorem:

Theorem 5.2. Let $P_d(x_1, x_2, \ldots, x_d) = x_1 \oplus \ldots x_{d-k} \oplus Q(x_{d-k+1}, \ldots, x_d)$, where Q is an arbitrary predicate of arity k and $k + 1 < \frac{d}{4}$. For all d large enough and all n large enough the random graph G with probability at least 0.85 has the following property. For every drunken algorithm $\mathcal{A}$, $\Pr_{y \leftarrow U_n}\{\Pr\{t^{\mathcal{A}}_{\Phi_{g(x)=g(y)}} > 2^{\Omega(n)}\} > 1 - 2^{-\Omega(n)}\} > 0.9$, where $t^{\mathcal{A}}_{\Phi}$ denotes the running time of the algorithm $\mathcal{A}$ on the formula Φ.

Proof. By the corollary 2.1 the random graph with probability 0.9 is a $(\frac{n}{10d}, d, \frac{1}{4}d)$-boundary expander. Theorem 5.1 implies that the average number of pairs x and y that $g(x) = g(y)$ is $2^{(1+2^{-\Omega(d)})n}$, where the averaging is on random graphs. The Markov inequality implies that with probability at least 0.95 for the random graph the number of pairs x and y such that $g(x) = g(y)$ is $2^{(1+2^{-\Omega(d)})n}$ (the constant is hidden in $\Omega(d)$). Therefore with probability at least 0.85 the random graph is a boundary expander and the upper bound on the number of pairs with equal values of g holds. We fix such graph G. The Markov inequality implies that for at least 0.9 fraction of strings $y \in \{0,1\}^n$ the following inequality $|g^{-1}(g(y))| < 2^{2^{-\Omega(d)}n}$ is satisfied. The predicate P contains at most k insensible variables (insensible variables are among $x_{d-k+1}, \ldots, x_d$), then Lemma 5.4 implies that the running time of any drunken algorithm on the formula $\Phi_{g(x)=g(y)}$ is at least $2^{\Omega(n)}$ with probability $1 - 2^{-\Omega(n)}$. □

Acknowledgement. The author thanks Ilya Mironov for bringing attention to [CEMT09], Edward A. Hirsch for fruitful discutions and also thanks Sofia Alexandrova, Anya Luter and Ilya Posov for useful comments that improved the readability of the paper.

References

[ABSRW00] M. Alekhnovich, E. Ben-Sasson, A. A. Razborov, and A. Wigderson. Pseudorandom generators in propositional proof complexity. In *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, page 43, Washington, DC, USA, 2000. IEEE Computer Society.

[AHI05] Michael Alekhnovich, Edward A. Hirsch, and Dmitry Itsykson. Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas. *J. Autom. Reason.*, 35(1-3):51–72, 2005.

[BSW01] E. Ben-Sasson and A. Wigderson. Short proofs are narrow — resolution made simple. *Journal of ACM*, 48(2):149–169, 2001.

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

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[EB05] Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. *Theory and Applications of Satisfiability Testing*, pages 61–75, 2005.

[ES03] Niklas Een and Niklas Sorensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518, 2003.

[Gol00] Oded Goldreich. Candidate one-way functions based on expander graphs. Technical Report 00-090, Electronic Colloquium on Computational Complexity, 2000.

[HLW06] S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43:439–561, 2006.

[MZ06] Ilya Mironov and Lintao Zhang. Applications of SAT solvers to cryptanalysis of hash functions. In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing—SAT 2006*, volume 4121 of *Lecture Notes in Computer Science*, pages 102–115, August 2006.

[NW94] N. Nisan and Avi Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.

[Tse68] G. S. Tseitin. On the complexity of derivation in the propositional calculus. *Zapiski nauchnykh seminarov LOMI*, 8:234–259, 1968. English translation of this volume: Consultants Bureau, N.Y., 1970, pp. 115–125.

[Urq87] A. Urquhart. Hard examples for resolution. *JACM*, 34(1):209–219, 1987.

A The bound on the number of solutions

In this section we estimate the number of pre-images of Goldreich's function based on the predicate $P_d(x_1, x_2, \ldots, x_d) = x_1 \oplus x_2 \oplus \cdots \oplus x_{d-k} \oplus Q(x_{d-k+1}, \ldots, x_d)$, where Q is an arbitrary predicate of arity k and $k + 1 < \frac{d}{4}$.

Theorem A.1 (cf. [CEMT09], theorem 4.1)**.** Let $P_d(x_1, x_2, \ldots, x_d) = x_1 \oplus x_2 \oplus \cdots \oplus x_{d-k} \oplus Q(x_{d-k+1}, \ldots, x_d)$, where Q is an arbitrary predicate of arity k and $k + 1 < \frac{d}{4}$. The graph G is obtained randomly in the following way: for every vertex in Y we choose independently at random d edges to X (repetitions are allowed). Then $\mathrm{E}[\#(x, y) \mid g(x) = g(y)] = 2^{(1+2^{-\Omega(d)})n}$, where g is a Goldreich's function based on G and the predicate P_d.

Proof. $\mathrm{E}[\#(x, y) \mid g(x) = g(y)] = \sum_{x,y \in \{0,1\}^n} \Pr\{g(x) = g(y)\}$.

Let $M \subseteq \{1, 2, \ldots, n\}$ be the set positions where x and y are different, we denote $m = |M|$. Since for every vertex in Y the edges of G are chosen independently, then $\Pr\{g(x) = g(y)\{= (\Pr\{(g(x))_1 = (g(y))_1\})^n$.

$\Pr\{(g(x))_1 = (g(y))_1\} = \Pr\{P_d(x_{i_1}, x_{i_2}, \ldots, x_{i_d}) = P_d(y_{i_1}, y_{i_2}, \ldots, y_{i_d})\}$, where $i_1, i_2, \ldots, i_d$ are independent random values that range between 1 and n with equal probability.

Since a predicate Q is unknown, we put down: $\Pr\{P_d(x_{i_1}, x_{i_2}, \ldots, x_{i_d}) = P_d(y_{i_1}, y_{i_2}, \ldots, y_{i_d})\} = \Pr\{P_d(x_{i_1}, x_{i_2}, \ldots, x_{i_d}) = P_d(y_{i_1}, y_{i_2}, \ldots, y_{i_d}) \mid Q(x_{i_{d-k+1}}, \ldots, x_{i_d}) = Q(y_{i_{d-k+1}}, \ldots, y_{i_d})\} \cdot q + \Pr\{P_d(x_{i_1}, x_{i_2}, \ldots, x_{i_d}) = P_d(y_{i_1}, y_{i_2}, \ldots, y_{i_d}) \mid Q(x_{i_{d-k+1}}, \ldots, x_{i_d}) \neq Q(y_{i_{d-k+1}}, \ldots, y_{i_d})\} \cdot (1 - q)$, where $q = \Pr\{Q(x_{i_{d-k+1}}, \ldots, x_{i_d}) = Q(y_{i_{d-k+1}}, \ldots, y_{i_d}\}$. Now we estimate each conditional probability separately. In the first case we estimate the probablity of the following event: the number of $i_1, i_2, \ldots, i_{d-k}$ that belongs to M is even. In the second case the last number should be odd. Consider a sequence p_j, where p_j is the probability that the number of $i_1, i_2, \ldots, i_j$ that belongs to M is even. The following recurrence relation follows from the independence of random variables: $p_{j+1} = \alpha(1-p_j)+(1-\alpha)p_j = \alpha+(1-2\alpha)p_j$, where $\alpha = \frac{m}{n}$, $p_0 = 1$. The solution of this recurrence relation is $p_j = \frac{1+(1-2\alpha)^j}{2}$, hence $1 - p_j = \frac{1-(1-2\alpha)^j}{2}$. Finally $\Pr\{f(x) = f(y)\} \leq \left(\frac{1+|1-2\alpha|^{d-k}}{2}\right)^n$.

$$\mathrm{E}[\#(x, y) \mid f(x) = f(y)] = \sum_{x \in \{0,1\}^n} \sum_{m=1}^{n} \sum_{y:\ \delta(x,y)=m} \Pr\{f(x) = f(y)\}$$
$$\leq 2^n \sum_k C_n^m \left(\frac{1 + |1 - 2\frac{k}{n}|^{d-k}}{2}\right)^n \leq n \cdot \max_{0 \leq \alpha \leq \frac{1}{2}} 2^{H(\alpha)n}(1 + (1 - 2\alpha)^{d-k})^n,$$

where $\delta(x, y)$ is the number of positions where x differs from y, $H(\alpha) = -\alpha \log \alpha - (1 - \alpha) \log(1 - \alpha)$ is a binary entropy, log is a binary logarithm.

Lemma A.1 ([CEMT09], from the proof of theorem 4.1)**.** There exists $\epsilon > 0$ such that for all $\alpha \in [0, \frac{1}{2}]$ and all big enough d the following inequality is satisfied $H(\alpha) + \log_2(1 + (1 - 2\alpha)^{d-k}) \leq 1 + 2^{-\epsilon d}$, where $k + 1 < \frac{d}{4}$.

Proof. We consider 4 cases depending on the α. We will choose values for the constants $\epsilon_1, \epsilon_2, \epsilon_3, \epsilon_4$ further in the proof.

Case 1: $\alpha > \epsilon_1$**.** $H(\alpha) + \log_2(1 + (1-2\alpha)^{d-k}) \overset{\ln(1+x)\le x}{\le} 1 + (1-2\epsilon_1)^{\frac{3d}{4}} / \ln 2 \le 1 + 2^{-\epsilon d}$, if $\epsilon < -\frac{3}{4}\log(1-2\epsilon_1)$ and d is big enough.

In the other cases $H(\alpha)$ is small. The following inequality follows from the Teylor decomposition of $\ln x$ in the point 2: $\ln(1+x) \le \ln 2 + \frac{x-1}{2}$. Then we may estimate:

$$\log(1 + (1-2\alpha)^{d-k}) \le 1 + \frac{(1-2\alpha)^{d-k} - 1}{2\ln 2} \overset{1+x\le e^x}{\le} 1 + \frac{e^{-2\alpha(d-k)} - 1}{2\ln 2} \le 1 + \frac{e^{-3\alpha\cdot d/2} - 1}{2\ln 2}$$

Case 2: $\epsilon_1 \ge \alpha > \epsilon_2/d$**.**

$$H(\alpha) + \log(1 + (1-2\alpha)^{d-k}) \le H(\epsilon_1) + 1 + \frac{e^{-3\epsilon_2/2} - 1}{2\ln 2} \le 1,$$

if we chose such a small ϵ_1 that $H(\epsilon_1) < \frac{1-e^{-3\epsilon_2/2}}{2\ln 2}$.

If $0 \le \alpha \le \frac{1}{2}$, then the inequality $(\alpha - 1)\log(1-\alpha) \le 2\alpha$ is satisfied because for $\alpha = 0$ the inequality becomes an equality, and the derivative of the difference between the left and the right sides is positive. We may estimate: $H(\alpha) \le \alpha \log \frac{1}{\alpha} + 2\alpha$.

For the remaining cases (where $\alpha \le \epsilon_2/d$) we choose $\epsilon_2 = \frac{1}{3}$, then $\frac{3}{2}\alpha d \le \frac{1}{2}$. For $-\frac{1}{2} \le x \le 0$ the inequality $e^x - 1 \le \frac{x}{2}$ is satisfied (it can be verified by considering a derivative). Now we may estimate:

$H(\alpha) + 1 + \frac{e^{-\frac{3}{2}\alpha d} - 1}{2\ln 2} \le (\alpha \log \frac{1}{\alpha} + 2\alpha) + 1 - \frac{3\alpha d}{8\ln 2} = 1 + \alpha(\log \frac{1}{\alpha} - \frac{3}{8\ln 2} \cdot d + 2)$

Case 3: $\epsilon_2/d \ge \alpha > 2^{-\epsilon_3 d}$**.** For $\epsilon_3 < \frac{3}{8\ln 2}$ and big enough d: $(\log \frac{1}{\alpha} - \frac{3}{8\ln 2} \cdot d + 2) < 0$.

Case 4: $2^{-\epsilon_3 d} \ge \alpha$**.** For $\epsilon < \epsilon_3$ and big enough d the following is satisfied: $\alpha \log \frac{1}{\alpha} \le \epsilon_3 d 2^{-\epsilon_3 d} \le 2^{-\epsilon d}$.

□

The statement of the theorem follows from the Lemma A.1. □