Electronic Colloquium on Computational Complexity, Report No. 41 (2004)

Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas

Michael Alekhnovich[*] Edward A. Hirsch[†] Dmitry Itsykson[‡]

February 24, 2004

Abstract

DPLL (for *Davis*, *Putnam*, *Logemann*, and *Loveland*) algorithms form the largest family of contemporary algorithms for SAT (the propositional satisfiability problem) and are widely used in applications. The recursion trees of DPLL algorithm executions on unsatisfiable formulas are equivalent to tree-like resolution proofs. Therefore, lower bounds for tree-like resolution (which are known since 1960s) apply to them.

However, these lower bounds say nothing about the behavior of such algorithms on satisfiable formulas. Proving exponential lower bounds for them in the most general setting would be equivalent to proving $\mathbf{P} \neq \mathbf{NP}$. In this paper, we give exponential lower bounds for two families of DPLL algorithms: *generalized myopic* algorithms (that read upto $n^{1-\epsilon}$ of clauses at each step and see the remaining part of the formula without negations) and *drunk* algorithms (that choose a variable using any complicated rule and then pick its value at random).

1 Introduction

SAT solving heuristics. The propositional satisfiability problem (*SAT*) is one of the most well-studied **NP**-complete problems. In this problem, one is asked whether a Boolean formula in conjunctive normal form (a conjunction of *clauses*, which are disjunctions of *literals*, which are variables or their negations) has an assignment that satisfies all its clauses. Despite the $\mathbf{P} \neq \mathbf{NP}$ conjecture, there is a lot of algorithms for SAT (motivated, in particular, by its importance for applications). DPLL algorithms (defined below) are based on the most popular approach that originates in the papers by Davis, Putnam, Logemann and Loveland [DP60, DLL62]. Very informally, these algorithms use a "divide-and-conquer" strategy: they split a formula into two subproblems by fixing a value of some literal, then they recursively process the arising formulas. These algorithms received much attention of researchers both from theory and practice and are heavily used in the applications.

Lower bounds for Resolution and the running time of DPLL algorithms. Propositional proof systems form one of the simplest and the most studied model in propositional calculus. Given

[*]Institute for Advanced Study, Princeton, US, `misha@ias.edu`.

[†]Steklov Institute of Mathematics, 27 Fontanka, St. Petersburg 191023, Russia. Email: `hirsch@pdmi.ras.ru`. Supported in part by Russian Science Support Foundation, RAS program of fundamental research "Research in principal areas of contemporary mathematics", RFBR grant #02-01-00089, and by Award No. RM1-2409-ST-02 of the U.S. Civilian Research & Development Foundation for the Independent States of the Former Soviet Union (CRDF).

[‡]St.Petersburg State University, St.Petersburg, Russia. Email: `dmitrits@mail.ru`.

ISSN 1433-8092

a formula F, a propositional proof system allows to show that F is unsatisfiable. For example, using the well-known *resolution rule* $\frac{x \vee C_1;\ \neg x \vee C_2}{C_1 \vee C_2}$ one can non-deterministically build a *resolution refutation of* F, which may be used as a certificate of unsatisfiability for the formula F. The size of the minimum tree-like resolution refutation and the running time of DPLL algorithms are related by the following well-known statement.

Fact 1.1. For each unsatisfiable formula the shortest tree-like resolution proof is at most polynomially longer than the smallest recursion tree of a DPLL algorithm, and vice versa.

Therefore, (sub)exponential lower bounds for tree-like resolution (starting with Tseitin's bounds [Tse68] and finishing with quite strong bounds of [PI00]) imply that any DPLL algorithm should work exponentially long proving that the corresponding formulas are unsatisfiable. However, these results say nothing in the case of *satisfiable* formulas. There are several reasons why the performance may differ on satisfiable and unsatisfiable instances:

- Experiments show that contemporary SAT solvers are able to solve much larger satisfiable formulas than unsatisfiable ones [SLBH02].
- Randomized algorithms fall out of scope, since they are not expected to prove unsatisfiability.
- If a DPLL algorithm is provably efficient (i.e. works in polynomial time) on some class of formulas, then one can interrupt the algorithm running on a formula from this class after sufficiently large number of steps if it has not found a satisfying assignment. This will result in a certificate of unsatisfiability that can be much smaller than the minimum tree-like resolution refutation.

Previously known lower bounds for satisfiable formulas. Despite the importance of this problem, only few works have addressed the question of the worst-case running time of SAT algorithms on satisfiable formulas. There has been two papers [Hir00, ABS02] on (specific) local search heuristics; as to DPLL algorithms it seems all we know are the bounds of [Nik02, ABM03, ABM04].

In the work of Nikolenko [Nik02] exponential lower bounds are proved for two specific DPLL algorithms (called `GUC` and `Randomized GUC`) on specially tailored satisfiable formulas.

Achlioptas, Beame, and Molloy [ABM03] prove the hardness of random formulas in 3-CNF with n variables and cn ($c < 4$) clauses for three specific DPLL algorithms (called `GUC`, `UC`, and `ORDERED-DLL`). It is an open problem to prove that these formulas are satisfiable (though it is widely believed they are). Recently, the same authors [ABM04] have proved an *unconditional* lower bound on satisfiable random formulas in 4-CNF for `ORDERED-DLL`. The latter result states that `ORDERED-DLL` takes exponential time with *constant* (rather than exponentially close to 1) probability.

Our contribution. Proving such bounds for DPLL algorithms in a greater generality is the ultimate goal of the present paper. We design two families of satisfiable formulas and show lower bounds for two general classes of algorithms (see Sect. 2.1 for precise definitions).

The first class of formulas simply encodes a linear system $Ax = b$ that has a unique solution over $\mathbb{GF}_2$, where A is a "good" expander. We prove that any *generalized myopic* DPLL algorithm that has a local access to the formula (i.e., can read upto $n^{1-\epsilon}$ clauses at every step) with high probability ought to make an exponential number of steps before it finds a satisfying assignment.

In our second result we describe a general way to cook a satisfiable formula out of any unsatisfiable formula hard for tree-like resolution so that the resulting formula is hard for any *drunk* DPLL algorithm that chooses a variable in an arbitrarily complicated way and then tries both its values in a random order.

Both classes of algorithms that we consider are classical DPLL backtracking algorithms, and in general are much less restricted than those studied before.

Organization of the paper. Section 2 contains basic notation and the rigorous definitions of DPLL algorithms that we consider. In the subsequent two sections we present our two main results. We discuss their possible extensions and open questions in Sect. 5.

2 Preliminaries

Let x be a Boolean variable, i.e., a variable that ranges over the set $\{0,1\}$. A *literal* of x is either x or $\neg x$. A *clause* is a disjunction of literals (considered as a set). A *formula* in this paper refers to a Boolean formula in conjunctive normal form, i.e., a conjunction of clauses (a formula is considered as a multiset). A formula in k*-CNF* contains clauses of size at most k. We will use the notation $\mathrm{Vars}(\Phi)$, $\mathrm{Vars}(Ax=b)$ to denote the set of variables occurring in a Boolean formula, in a system of equations, etc.

An *elementary substitution* $v := \varepsilon$ just points a variable v and a Boolean value $\varepsilon \in \{0,1\}$. A *substitution* (also called a *partial assignment*) is a set of elementary substitutions for different variables. The result of applying a substitution ρ to a formula F (denoted by $F[\rho]$) is a new formula obtained from F by removing the clauses containing literals satisfied by ρ and removing the opposite literals from other clauses.

We say that an assignment α *satisfies* a Boolean function f if $f(\alpha)=1$. For Boolean functions $f_1,\ldots,f_k,g$ we say that $f_1,\ldots,f_k$ *semantically imply* g, (denoted $f_1,\ldots,f_k \models g$), if every assignment to the variables in $V = \mathrm{Vars}(f_1)\cup\ldots\cup\mathrm{Vars}(f_k)\cup\mathrm{Vars}(g)$ satisfying $f_1,\ldots,f_k$, satisfies g as well (i.e. $\forall\alpha\in\{0,1\}^V(f_1(\alpha)=\cdots=f_k(\alpha)=1\Rightarrow g(\alpha)=1)$).

For a non-negative integer n, let $[n]=\{1,2,\ldots,n\}$. For a vector $v=(v_1,...,v_m)$ and index set $I\subseteq[m]$ we denote by v_I the subvector with coordinates chosen according to I. For a matrix A and a set of rows $I\subseteq[m]$ we use the notation A_I for the submatrix of A corresponding to these rows. In particular, we denote the ith row of A by A_i and identify it with the set $\{j \mid A_{ij}=1\}$. The cardinality of this set is denoted by $|A_i|$.

2.1 DPLL algorithms: general setting

A DPLL algorithm is a recursive algorithm. At each step, it simplifies the input formula F (without affecting its satisfiability), chooses a variable v in it and makes two recursive calls for the formulas $F[v:=1]$ and $F[v:=0]$ in some order; it outputs "Satisfiable" iff at least one of the recursive calls says so (note that there is no reason to make the second call if the first one was successful). The recursion proceeds until the formula trivializes, i.e., it becomes empty (hence, satisfiable) or one of the clauses becomes empty (hence, the formula is unsatisfiable).

A DPLL algorithm is determined by its simplification rules and two heuristics: Heuristic A that chooses a variable and Heuristic B that chooses its value to be examined first. A formal description is given in Fig. 1. Note that if $\mathbf{P}=\mathbf{NP}$ and Heuristic B is not restricted, it can simply choose the correct values and the algorithm will terminate quickly. Therefore, in order to prove unconditional lower bounds one has to restrict the simplification rules and heuristics and prove the result for the restricted model. In this paper, we consider two models: generalized myopic algorithms and drunk algorithms. Both models extend the original algorithm of [DLL62], which uses the unit clause and pure literal rules and no non-trivial Heuristics A and B.

```
Algorithm $\mathcal{A}$.
Input: formula $F$ in CNF.
Output: “Satisfiable” or “Unsatisfiable”.

  1. Simplify $F$ using simplification rules.
  2. If $F$ is empty, return “Satisfiable”.
  3. If $F$ contains the empty clause, return “Unsatisfiable”.
  4. Choose a variable $v$ using Heuristic A.
  5. Choose a Boolean value $\varepsilon$ using Heuristic B.
  6. If $\mathcal{A}(F[v := \varepsilon])$ returns “Satisfiable”, return “Satisfiable”.
  7. If $\mathcal{A}(F[v := \neg\varepsilon])$ returns “Satisfiable”, return “Satisfiable”.
  8. Return “Unsatisfiable”.
```

Figure 1: A DPLL algorithm.

Drunk algorithms. Heuristic A of a drunk algorithm can be arbitrarily complicated (even non-recursive). This is compensated by the simplicity of Heuristic B: it chooses 0 or 1 at random. The simplification rules are

Unit clause elimination. If the formula F contains a clause that consists of a single literal l, replace F by $F[l := 1]$, where $l := 1$ denotes the elementary substitution that satisfies the literal l.

Pure literal elimination. If the formula F contains a literal l such that its negation does not occur in any clause[1], replace F by $F[l := 1]$.

Subsumption. If the formula F contains a clause that contains another clause as a subset, delete the larger clause.

Note that `Randomized GUC` with pure literal elimination considered in [Nik02] is a drunk algorithm (that does not use subsumption).

In Section 4 we prove an exponential lower bound on the running time of drunk algorithms on satisfiable formulas obtained by a simple construction that uses (known) hard unsatisfiable formulas.

Myopic algorithms. Both heuristics are restricted w.r.t. the parts of formula that they can read (this can be viewed as accessing the formula via an oracle). Heuristic A can read

- $K(n)$ clauses of the formula (where n is the number of variables in the original input formula and $K(n) = n^{1-\epsilon}$ is a function with $\epsilon > 0$);
- the formula with negation signs removed;

[1] An occurrence of a positive literal is an occurrence of the corresponding variable *without* the negation.

- the number of occurrences of each literal.

Heuristic B may use the information obtained by Heuristic A. The information revealed about the formula can be used in the subsequent recursive calls (but not in other branches of the recursion tree).

The only simplification rule is pure literal elimination. Also the unit clause elimination can be easily implemented by choosing the proper variable and value. In particular, heuristics `ORDERED-DLL`, `GUC` and `UC` considered in [ABM03] yield generalized myopic algorithms. Note that our definition generalizes the notion of myopic algorithms introduced in [AS00].

Formally, the heuristics are unable to read all clauses containing a variable if this variable is too frequent. However, it is easy to see that we can restrict our hard formulas (that we use for proving our exponential lower bound) so that every variable occurs $O(\log n)$ times, see Remark 3.1.

In Section 3 we prove an exponential lower bound on the running time of myopic algorithms on satisfiable formulas based on expanders.

2.2 DPLL recursion tree

DPLL *recursion tree* is a binary tree (a node may have zero, one, or two children) in which nodes correspond to the intermediate subproblems that arise after the algorithm makes a substitution, edges correspond to the recursive calls on the resulting formulas. The computation of a DPLL algorithm thus can be considered as depth-first traverse of the recursion tree from the left to the right; in particular, the rightmost leaf always corresponds to the satisfying assignment (if any), the overall running time is proportional to the size of the tree.

For a node v in the computation tree by ρ_v we denote the partial assignment that was set prior to visiting v, thus the algorithm at v works on the subformula $F[\rho_v]$.

2.3 Expanders

An expander is a bounded-degree graph that has many neighbors for every subset of its nodes. Similarly to [ABSRW00], we use a more general notion of expander as an $m \times n$ matrix. There are two notions of expanders: expanders and boundary expanders. The latter notion is stronger as it requires the existence of unique neighbors. However, every good expander is also a boundary expander.

Definition 2.1. For a set of rows $I \subseteq [m]$ of an $m \times n$ matrix A, we define its *boundary* $\partial_A I$ (or just ∂I) as the set of all $j \in [n]$ (called *boundary elements*) such that there exists exactly one row $i \in I$ that contains j. We say that A is an (r, s, c)*-boundary expander* if

1. $|A_i| \le s$ for all $i \in [m]$, and
2. $\forall I \subseteq [m]\ (|I| \le r \Rightarrow |\partial I| \ge c \cdot |I|)$.

Matrix A is an (r, s, c)*-expander* if condition 2 is replaced by

2′. $\forall I \subseteq [m]\ (|I| \le r \Rightarrow |\bigcup_{i \in I} A_i| \ge c \cdot |I|)$.

We define the boundary and boundary elements of equation(s) in a linear system $Ax = b$ similarly to those of rows in a matrix A.

Lemma 2.1. Any $(r, 3, c)$-expander is an $(r, 3, 2c - 3)$-boundary expander.

Proof. Assume that A is $(r, 3, c)$-expander. Consider a set of its rows I with $|I| \le r$. Since A is an expander $|\bigcup_{i\in I} A_i| \ge c|I|$. On the other hand we may estimate separately the number of boundary and non-boundary variables which will give $|\bigcup_{i\in I} A_i| \le E + (3|I| - E)/2$, where E is the number of boundary variables. This implies $E + (3|I| - E)/2 \ge c|I|$ and $E > (2c - 3)|I|$. $\square$

3 An exponential lower bound for myopic algorithms

In this section, we prove an exponential lower bound on the running time of generalized myopic algorithms (described in Sect. 2.1) on satisfiable formulas. The proof strategy is as follows: we take a full-rank $n \times n$ 0/1-matrix A having certain expansion properties and construct a uniquely satisfiable Boolean formula Φ expressing the statement $Ax = b$ (modulo 2) for some vector b. Then we prove that if one obtains an unsatisfiable formula from Φ using a reasonable substitution, the resulting formula is hard for tree-like resolution (the proof is similar to that of [BSW01]). Finally, we show that changing several bits in the vector b, while changes the satisfying assignment, does not affect the behavior of generalized myopic algorithm that did not reveal these bits, which implies it encounters a hard unsatisfiable formula on its way to the satisfying assignment.

In what follows, we prove the existence of appropriate expanders (Sect. 3.1) and examine their properties (Sect. 3.2). Then we give the construction of the corresponding Boolean formulas (Sect. 3.3) and prove the statement concerning the behavior of a generalized myopic algorithm on unsatisfiable formulas (Sect. 3.4). Finally, we prove our main result of this section (Sect. 3.5).

3.1 The existence of appropriate expanders

We now prove the existence of expanders that we use to construct satisfiable formulas hard for myopic DPLL algorithms.

Theorem 3.1. For every sufficiently large n, there exists an $n \times n$ non-degenerate matrix $A^{(n)}$ such that $A^{(n)}$ is an $(n/\log^{14} n, 3, 25/13)$-expander.

Let $\binom{[n]}{3}$ be the set of all $\{0,1\}^n$-vectors of Hamming weight 3 (i.e., containing exactly three 1's). We use a probabilistic construction: the rows of a larger matrix are drawn at random from the set of all vectors of Hamming weight 3; then we choose a submatrix of the appropriate size. In order to establish the goal, we prove two lemmas.

Lemma 3.1. Let A be a $\Delta n \times n$ matrix (Δ may depend on n) in which each row is randomly chosen from $\binom{[n]}{3}$. Assume that $c < 2$ is a constant and $r = o\left(\frac{n}{\Delta^{1/(2-c)}}\right)$. Then with probability $1 - o(1)$ the matrix A is an $(r, 3, c)$-expander.

Proof. The probability p_t of the event that there exists a subset of rows I of size $t \le r$ and a subset of columns $J \supseteq A_I$ of size $\lfloor ct \rfloor$ is upper bounded as

$$p_t \le \binom{\Delta n}{t}\binom{n}{\lfloor ct\rfloor}\left(\frac{ct}{n}\right)^{3t} \le \left(\frac{e\Delta n}{t}\right)^t \left(\frac{en}{ct}\right)^{ct}\left(\frac{ct}{n}\right)^{3t} = \left[e^{1+c}c^{3-c}\Delta\left(\frac{n}{t}\right)^{c-2}\right]^t \le$$
$$\le \left[e^{(1+c)/(2-c)}c^{(3-c)/(2-c)}r\frac{\Delta^{1/(2-c)}}{n}\right]^{(2-c)t}.$$

Clearly, $p_1 = o(1)$. Since for sufficiently large n, $\sum_{t=1}^{r} p_t \le 2p_1$, the lemma follows. $\square$

Lemma 3.2. Let L be a linear subspace of $\{0,1\}^n$ of codimension k. Let vector v be chosen uniformly at random from $\binom{[n]}{3}$. Then $\Pr[v \notin L] = \Omega(\frac{k}{n})$.

Proof. L can be specified as a kernel of a $k \times n$ matrix M of full rank (i.e., $L = \{u \mid Mu = 0\}$). The product Mv is distributed as a sum of three columns randomly chosen (without replacement) from the matrix M; we need to estimate the probability that this sum equals zero. Let $M_{i_1}, M_{i_2}, M_{i_3}$ be the three randomly chosen columns of M.

Case 1: $k \geq 3$. In this case, consider the vector $u = M_{i_1} + M_{i_2}$. Since $\operatorname{rk} M = k$, there are at least $k-2$ other columns in M different from u. Thus, $M_{i_3} \neq u$ with probability at least $\frac{k-2}{n}$.

Case 2: $k < 3$.

Case 2a: $\exists j_1 j_2\ \forall j\ (j \notin \{j_1, j_2\} \Rightarrow M_j = M_{j_1} + M_{j_2})$. Since $\operatorname{rk} M > 0$, either M_{i_1} or M_{i_2} is nonzero. With probability $1/n$ the nonzero column is chosen as the first column. If this happens, then with probability at least $\frac{n-2}{n-1} \cdot \frac{n-3}{n-2}$ the second and the third column are chosen from those equal to $M_{i_1} + M_{i_2}$. Thus, with probability at least $\frac{1}{n} \cdot \frac{n-2}{n-1} \cdot \frac{n-3}{n-2} \geq \frac{1}{2n}$ $M_{i_1} + M_{i_2} + M_{i_3} \neq 0$.

Case 2b: The condition of case 2a does not hold. Consider the vector $u = M_{i_1} + M_{i_2}$. By our assumption, there is at least one column $j \notin \{i_1, i_2\}$ different from u. With probability at least $\frac{1}{n-2}$ this column will be chosen as the third one.

□

Proof of Theorem 3.1. The estimation of the number Δn of random vectors that suffices to obtain a $\Delta n \times n$ matrix of full rank resembles the analysis of the well-known "Coupon Collector" problem. Let $S_0 = \emptyset$, $S_{i+1} = S_i \cup \{v_i\}, v_i \in_U \binom{[n]}{3}$. Let T be the first step when the vector system S_T is complete. It is easy to see that the expectation of T is $O(n \log n)$: Lemma 3.2 shows that if the dimension of $\operatorname{Span}(S_k)$ is t then $\dim \operatorname{Span}(S_{k+1}) = t + 1$ with probability $\Omega\left(\frac{n-t}{n}\right)$. Thus $O(\frac{n}{n-t})$ steps suffice on average to increase the dimension from t to $t+1$. By linearity of expectation,

$$\mathbf{E}T \leq O\left(\frac{n}{n} + \frac{n}{n-1} + \frac{n}{n-2} + \ldots + \frac{n}{1}\right) = O(n \log n).$$

Let a' be the constant in the $O(\cdot)$ notation above, i.e., $\mathbf{E}T \leq a' n \log n$. Let $a'' = \frac{a'}{\epsilon}$ (we will choose ϵ later). By Markov inequality,

$$\Pr\{T > a'' n \log n\} < \epsilon.$$

Let us choose ϵ and ϵ' so that $\epsilon + \epsilon' < 1$. For sufficiently large n, Lemma 3.1 guarantees that A is an $(n/\log^{14} n, 3, 25/13)$-expander with probability at least $1 - \epsilon'$. By the above reasoning, also $\operatorname{rk} A = n$ with a positive probability. Thus, we can choose n linear independent rows of A; the resulting $n \times n$ matrix is an $(n/\log^{14} n, 3, 25/13)$-expander. □

Remark 3.1. It is easy to see that one can add an additional requirement: for every column j, there is only $O(\log n)$ rows A_i such that $A_{ij} = 1$. Using such expanders would result in hard formulas with only $O(\log n)$ occurrences of every variable.

3.2 Closure operators

Throughout this section, A denotes an $(r, 3, c)$-boundary expander. We need two operations of taking closure of a set of columns w.r.t. matrix A. The first was defined in [AR01].

Definition 3.1. Let $A \in \{0,1\}^{m \times n}$. For a set of columns $J \subseteq [n]$ define the following inference relation $\vdash_J$ on the sets $[m]$ of rows of A:

$$I \vdash_J I_1 \iff |I_1| \le r/2 \ \wedge\ \partial_A(I_1) \subseteq \left[\bigcup_{i \in I} A_i \cup J\right]. \tag{3.1}$$

That is, we allow to derive rows of A from already derived rows. We can use these derived rows in further derivations (for example, derive new rows from $I \cup I_1$). Let the *closure* $\mathrm{Cl}(J)$ of J be the set of all rows which can be inferred via $\vdash_J$ from the empty set.

The following lemma was proved in [AR01, Lemma 3.16].

Lemma 3.3. For any set J with $|J| \le (cr/2)$, $|\mathrm{Cl}(J)| \le r/2$.

We also need another (stronger) closure operation the intuitive sense of which is to extract a good expander out of a given matrix by removing rows and columns.

Definition 3.2. For an $A \in \{0,1\}^{m \times n}$ and a subset of its columns $J \subseteq [n]$ we define an inference relation $\vdash'_J$ on subsets of rows of A:

$$I \vdash'_J I_1 \iff |I_1| \le r/2 \ \wedge\ \left|\partial_A(I_1) \setminus \left[\bigcup_{i \in I} A_i \cup J\right]\right| < c/2|I_1| \tag{3.2}$$

Given a set of rows I and a set of columns J consider the following *cleaning* step:

- If there exists a nonempty subset of rows I_1 such that $I \vdash'_J I_1$, then
  - Add I_1 to I.
  - Remove all rows corresponding to I_1 from A.

Repeat the cleaning step as long as it is applicable. Fix any particular order on the sets to exclude ambiguity, initialize $I = \emptyset$ and denote the resulting content of I at the end by $\mathrm{Cl}^{\mathrm{e}}(J)$.

Lemma 3.4. Assume that A is an arbitrary matrix and J is a set of its columns. Let $I' = \mathrm{Cl}^{\mathrm{e}}(J)$, $J' = \bigcup_{i \in \mathrm{Cl}^{\mathrm{e}}(J)} A_i$. Denote by $\hat{A}$ the matrix that results from A by removing the rows corresponding to I' and columns to J'. If $\hat{A}$ is non-empty than it is an $(r/2, 3, c/2)$-boundary expander.

Proof. Follows immediately from the definition of Cl^{e}. □

Lemma 3.5. If $|J| < cr/4$, then $|\mathrm{Cl}^{\mathrm{e}}(J)| < 2c^{-1}|J|$.

Proof. Assume that $|\mathrm{Cl}^{\mathrm{e}}(J)| \ge 2c^{-1}|J|$. Consider the sequence $I_1, I_2, \ldots, I_t$ appearing in the cleaning procedure; i.e.,

$$I_1 \cup I_2 \cup \ldots \cup I_k \vdash'_J I_{k+1}.$$

Note that $I_i \cap I_{i'} = \emptyset$ for $i \ne i'$, because we remove the implied set of rows from A at each cleaning step. Denote by $C_t = \bigcup_{k=1}^{t} I_k$ the set of rows derived in t steps.

Let T be the first t such that $|C_t| \geq 2c^{-1}|J|$. Note that $|C_T| \leq 2c^{-1}|J| + r/2 \leq r$, hence $|J| < cr/4 \leq c|C_T|/4$. Because of the expansion properties of A, $\partial C_T \geq c|C_T|$, which implies

$$|\partial C_T \setminus J| \geq c|C_T| - |J| > c|C_T|/2. \tag{3.3}$$

On the other hand, every time we add some I_{t+1} to C_t during the cleaning procedure, only $c/2|I_{t+1}|$ new elements can be added to $\partial C_t \setminus J$ (of those elements that have never been there before). This implies

$$|\partial C_T \setminus J| \leq c|C_T|/2,$$

which contradicts (3.3). □

3.3 Hard formulas based on expanders

Let A be an $n \times n$ matrix provided by Theorem 3.1, let also $r = n/\log^{14} n$, $c' = 25/13$ be the parameters of the theorem. Denote $c = 2c' - 3$ (thus A is an $(r, 3, c)$-boundary expander).

Definition 3.3. Let b is a vector from $\{0,1\}^n$. Then $\Phi(b)$ is the formula expressing the equality $Ax = b$ (modulo 2), namely, every equation $a_{ij_1}x_{j_1} + a_{ij_2}x_{j_2} + a_{ij_3}x_{j_3} = b_i$ is transformed into the 4 clauses on $x_{j_1}, x_{j_2}, x_{j_3}$ satisfying all its solutions. Sometimes we identify an equation with the corresponding clauses.

Remark 3.2. The formula $\Phi(b)$ has several nice properties that we use in our proofs. First, note that $\Phi(b)$ has exactly one satisfying assignment (since $\operatorname{rk} A = n$). It is also clear that a myopic DPLL algorithm has no reasonable chance to apply pure literal elimination to it, because for any substitution ρ, the formula $\Phi(b)[\rho]$ never contains a pure literal unless this pure literal is contained in a unit clause. Moreover, the number of occurrences of a literal in $\Phi(b)[\rho]$ always equals the number of occurrences of the opposite literal (recall that a formula is a *multi*set of clauses); again the only exception is literals occurring in unit clauses.

To the abuse of notation we identify $j \in J$ (where J is a set of columns of A) with the variable x_j.

3.4 Behavior of myopic algorithms on unsatisfiable formulas

Definition 3.4. A substitution ρ is said to be *locally consistent* w.r.t. the linear system $Ax = b$ if and only if ρ can be extended to an assignment on X which satisfies the equations corresponding to $\mathrm{Cl}(\rho)$:

$$A_{\mathrm{Cl}(\rho)}x = b_{\mathrm{Cl}(\rho)}.$$

Lemma 3.6. Assume that A is $(r, 3, c)$-boundary expander, Let $b \in \{0,1\}^m$ and ρ is a locally consistent partial assignment. Then for any set $I \subset [m]$ with $|I| \leq r/2$, ρ can be extended to an assignment x which satisfies the subsystem $A_I x = b_I$.

Proof. Assume for the contradiction that there exists set I for which ρ cannot be extended to satisfy $A_I x = b_I$; choose the minimal such I. Then $\partial_A(I) \subseteq \mathrm{Vars}(\rho)$, otherwise one could remove an equation with boundary variable in $\partial_A(I) \setminus \mathrm{Vars}(\rho)$ from I. Thus, $\mathrm{Cl}(\rho) \supseteq I$, which contradicts Definition 3.4. □

The *width* [BSW01] of a resolution proof is the maximal length of a clause in the proof. We need the following lemma which is a straightforward generalization of [BSW01, Theorem 4.4].

Lemma 3.7. For any matrix A which is an $(r,3,c)$-boundary expander and any vector $b \notin \mathrm{Im}(A)$ any resolution proof of the system

$$Ax = b \tag{3.4}$$

must have width at least $cr/2$.

Proof. For a clause C define Ben-Sasson–Wigderson measure as

$$\mu(C) = \min_{(A_I x = b_I) \models C} |I|.$$

Similarly to the proof of [BSW01, Theorem 4.4], μ is a subadditive measure, for any D appearing in the translation[2] of (3.4) to CNF $\mu(D) = 1$ and $\mu(\emptyset) \geq r$ (the latter inequality follows from the fact that any set I' ($I' \models \emptyset$) with $|I'| < r$ has a non-empty boundary, and an equality containing a boundary variable can be removed from the subsystem $A_{I'}x = b_{I'}$ leaving it still contradictory).

It follows that any resolution refutation of the system (3.4) contains a clause C s.t. $r/2 \leq \mu(C) < r$. Consider a minimal I s.t. $(A_I x = b_I) \models C$. As in [BSW01] we claim that C has to contain all variables corresponding to $\partial_A(I)$. Indeed, if there exists a boundary variable in the equation $A_i x = b_i$ ($i \in I$) not included in C then we may remove this equation so that $(A_{[I\setminus i]}x = b_{[I\setminus i]}) \models C$. Thus, C contains all boundary variables of I and there are at least $c|I| \geq cr/2$ of them. □

We also need the following lemma from [BSW01]:

Lemma 3.8 ([BSW01, Corollary 3.4]). The size of any tree-like resolution refutation of a 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 length of a clause in Ψ.

Lemma 3.9. If a locally consistent substitution ρ s.t. $|\mathrm{Vars}(\rho)| \leq cr/4$ results in an unsatisfiable formula $\Phi(b)[\rho]$ then every generalized myopic DPLL algorithm would work $2^{\Omega(r)}$ on $\Phi(b)[\rho]$.

Proof. The work of any DPLL algorithm on an unsatisfiable formula can be translated to tree-like resolution refutation so that the size of the refutation is the working time of the algorithm. Thus, it is sufficient to show that the minimal tree-like resolution refutation size of $\Phi(b)[\rho]$ is large.

Denote by $I = \mathrm{Cl}^{\mathrm{e}}(\rho)$, $J = \bigcup_{i \in I} A_i$. By Lemma 3.5 $|I| \leq r/2$. By Lemma 3.6 ρ can be extended to another partial assignment ρ' on variables x_J, s.t. ρ' satisfies every linear equation in $A_I x = b_I$. The restricted formula $(Ax = b)|_{\rho'}$ still encodes an unsatisfiable linear system, $A'x = b'$, where matrix A' results from A by removing rows corresponding to I and variables corresponding to J. By Lemma 3.4, A' is an $(r/2, 3, c/2)$-boundary expander. Lemmas 3.7 and 3.8 now imply that the minimal tree-like resolution refutation of the Boolean formula corresponding to the system $A'x = b'$ has size $2^{\Omega(r)}$. □

3.5 Behavior of myopic algorithms on satisfiable formulas

We fix A, r, c, c' of Sect. 3.3 and $m = m(n) = n$ throughout this section.

Theorem 3.2. For every deterministic generalized myopic DPLL algorithm $\mathcal{A}$ that reads at most $K = K(n)$ clauses per step, $\mathcal{A}$ stops on $\Phi(b)$ in $2^{o(r)}$ steps with probability $2^{-\Omega(r/K)}$. The probability is taken over b uniformly distributed on $\{0,1\}^n$.

[2]See Definition 3.3.

Corollary 3.1. Let $\mathcal{A}$ be any (randomized) generalized myopic DPLL algorithm that reads at most $K = K(n)$ clauses per step. $\mathcal{A}$ stops on $\Phi(b)$ (a satisfiable formula in 3-CNF containing n variables and $4n$ clauses, described in Sect. 3.3) in $2^{o(n \log^{-14} n)}$ steps with probability $2^{-\Omega(K^{-1} n \log^{-14} n)}$ (taken over random bits used by the algorithm and over b uniformly distributed on $\{0,1\}^n$).

Proof of Theorem 3.2. The proof strategy is to show that during its very first steps the algorithm does not get enough information to guess a correct substitution with non-negligible probability. Therefore, the algorithm chooses an incorrect substitution and has to examine an exponential-size subtree by Lemma 3.9.

Without loss of generality, we assume that our algorithm is a *clever myopic* algorithm. We define a clever myopic algorithm w.r.t. matrix A as a generalized myopic algorithm (defined as in Section 2.1) that

- has the following ability: whenever it reveals occurrences of the variables x_J (at least one entry of each) it can also read all clauses in $\text{Cl}(J)$ for free and reveal the corresponding occurrences;
- never asks for a number of occurrences of a literal (syntactical properties of our formula imply that $\mathcal{A}$ can compute this number itself: the number of occurrences outside unit clauses does not depend on the substitutions that $\mathcal{A}$ has made; all unit clauses belong to $\text{Cl}(J)$);
- always selects one of the revealed variables;
- never makes stupid moves: whenever it reveals the clauses $\vec{C}$ and chooses the variable x_j for branching it makes the right assignment $x_j = \epsilon$ in the case when $\vec{C}$ semantically imply $x_j = \epsilon$ (this assumption can only save the running time).

Proposition 3.1. After the first $\lfloor \frac{cr}{6K} \rfloor$ steps a clever myopic algorithm reads at most $r/2$ bits of b.

Proof. At each step the algorithm makes K clause queries, asking for $3K$ variable entries. This will sum up to $3K(cr/(6K))$ variables which will result by Lemma 3.3 in at most $r/2$ revealed bits of b. □

Recall that an assignment ρ is locally consistent if it can be extended to an assignment that satisfies $A_{\text{Cl}(\rho)} x = b_{\text{Cl}(\rho)}$.

Proposition 3.2. During the first $\lfloor \frac{cr}{6K} \rfloor$ steps the current partial assignment made by a clever myopic algorithm is locally consistent (in particular, the algorithms does not backtrack).

Proof. The statement follows by repeated application of Lemma 3.6. Note that the definition of clever myopic algorithm requires that it chooses a locally consistent assignment if possible.

Formally we prove the proposition by induction. In the beginning of the execution the current partial assignment is empty, hence it is locally consistent. By the definition of a clever myopic algorithm, whenever it makes a step t (where $t < \lfloor \frac{cr}{6K} \rfloor$) having a locally consistent partial assignment ρ_t it extends this assignment to an assignment ρ_{t+1} that is also locally consistent if possible. By Lemma 3.6 it can always do so as long as $|\text{Cl}(\text{Vars}(\rho_t) \cup \{x_j\})| \le r/2$ for the newly chosen variable x_j. □

Assume now that b chosen at random is hidden from $\mathcal{A}$. Whenever an algorithm reads the information about a clause corresponding to the linear equation $A_i x = b_i$ it reveals the ith bit of b. Let us observe the situtation after the first $\lfloor \frac{cr}{6K} \rfloor$ steps of $\mathcal{A}$, i.e., the $\lfloor \frac{cr}{6K} \rfloor$-th vertex v in the leftmost branch in the DPLL tree of the execution of $\mathcal{A}$. By Proposition 3.1 the algorithm reads at most $r/2$ bits of b. Denote by $I_v \subset [m]$ the set of the revealed bits, and by R_v the set of the assigned variables, $|R_v| = \lfloor \frac{cr}{6K} \rfloor$. The idea of the proof is that $\mathcal{A}$ cannot guess the true values of x_{R_v} by observing only r bits of b. Denote by ρ_v the partial assignment to the variables in R_v made by $\mathcal{A}$. Consider the following event

$$E = \{(A^{-1}b)_{R_v} = \rho_v\}$$

(recall that our probability space is defined by the 2^m possible values of b). This event holds if and only if the formula $\Phi(b)|_{\rho_v}$ is satisfiable. For $I \subset [m], R \subset [n], \vec{\epsilon} \in \{0,1\}^I, \rho \in \{0,1\}^R$ we want to estimate the conditional probability

$$\Pr[E \mid I_v = I,\ R_v = R,\ b_{I_v} = \vec{\epsilon},\ \rho_v = \rho\,]. \tag{3.5}$$

If we show that this conditional probability is small (irrespectively of the choice of I, R, $\vec{\epsilon}$, and ρ), it will follow that the probability of E is small.

We use the following lemma (and delay its proof for a moment).

Lemma 3.10. Assume that an $m \times n$ matrix A is an $(r, 3, c')$-expander, $X = \{x_1, \ldots, x_n\}$ is a set of variables, $\hat{X} \subseteq X$, $|\hat{X}| < r$, $b \in \{0,1\}^m$, and $\mathcal{L} = \{\ell_1, \ldots, \ell_k\}$ (where $k < r$) is a tuple of linear equations from the system $Ax = b$. Denote by L the set of assignments to the variables in $\hat{X}$ that can be extended on X to satisfy $\mathcal{L}$. If L is not empty then it is an affine subspace of $\{0,1\}^{\hat{X}}$ of dimension greater than $|\hat{X}|\left(\frac{1}{2} - \frac{14-7c'}{2(2c'-3)}\right)$.

Choose $\mathcal{L} = \{A_i x = \epsilon_i\}_{i \in I}$, $X = \mathrm{Vars}(\mathcal{L})$, $\hat{X} = R$, $|\hat{X}| = \lfloor cr/(6K) \rfloor$, recall that $c' = 25/13$. Then Lemma 3.10 says that $\dim L > \frac{2}{11}|R|$, where L is the set of locally consistent assignments to the variables in R. Let

$$(\hat{b})_i = \begin{cases} \epsilon_i, & i \in I, \\ b_i, & \text{otherwise} \end{cases}.$$

Note that $\hat{b}$ has the distribution of b when we fix $I_v = I$ and $b_I = \vec{\epsilon}$. The vector $\hat{b}$ is independent from the event $E_1 = [I_v = I \wedge R_v = R \wedge b_{I_v} = \vec{\epsilon} \wedge \rho_v = \rho]$. This is because in order to determine whether E_1 holds it is sufficient to observe the bits b_I only. Clearly, $(A^{-1}\hat{b})_R$ is distributed uniformly on L (note that A is a bijection), thus

$$\begin{aligned}
&\Pr[E \mid I_v = I,\ R_v = R,\ b_{I_v} = \vec{\epsilon},\ \rho_v = \rho\,] \\
&= \Pr[(A^{-1}\hat{b})_R = \rho \mid I_v = I,\ R_v = R,\ b_{I_v} = \vec{\epsilon},\ \rho_v = \rho\,] \\
&= \Pr[(A^{-1}\hat{b})_R = \rho\,] \\
&\le 2^{-\dim L} < 2^{-\frac{2}{11}|R|} \le 2^{-\frac{cr}{1000K}}.
\end{aligned}$$

However, if E does not happen then by Lemma 3.9 it takes time $2^{\Omega(r)}$ for $\mathcal{A}$ to refute the resulting unsatisfiable system (note that by Proposition 3.2 the assignment ρ_v is locally consistent). □

Proof of Lemma 3.10. First we repeatedly eliminate variables and equations from $\mathcal{L}$ until we get rid of

(1) equations containing boundary variables not from $\hat{X}$;

(2) equations containing more than one boundary variable.

This is done by the repetition of the following two procedures (in any order) as long as at least one of them is applicable:

Procedure 3.1. If $\mathcal{L}$ contains an equation ℓ with boundary element $j \in \partial\mathcal{L}$ s.t. $x_j \notin \hat{X}$, then remove ℓ from $\mathcal{L}$.

Note that Procedure 3.1 does not change L and $\hat{X}$. Therefore, if the claim of our lemma holds for the new system and new $\hat{X}$, it holds for the original one as well.

Procedure 3.2. If $\mathcal{L}$ contains an equation ℓ with at least two boundary elements j_1, j_2 s.t. $x_{j_1}, x_{j_2} \in \hat{X}$, then remove ℓ from $\mathcal{L}$ and all these (two or three) boundary elements from $\hat{X}$.

This procedure decreases $|\hat{X}|$ by 2 (or by 3) and decreases $\dim L$ by 1 (resp., by 2). Therefore, if the claim of our lemma holds for the new system and new $\hat{X}$, it holds for the original one as well.

Thus, it is enough to prove the claim of our lemma for the case where none of the procedures above is applicable to $\mathcal{L}$. Then $\partial\mathcal{L}$ is covered by $\hat{X}$; in particular,

$$k(2c'-3) \le |\partial\mathcal{L}| \le |\hat{X}|,$$

which implies

$$k \le \frac{|\hat{X}|}{2c'-3}. \tag{3.6}$$

(Note that we have used Lemma 2.1 here.) Denote by $\mathcal{L}' \subseteq \mathcal{L}$ the subset of equations that contain at least one variable from $\hat{X}$. Since none of them contains two boundary variables, and there are at least $k(2c'-3)$ such boundary variables,

$$|\mathcal{L}'| \ge k(2c'-3).$$

Let $\bar{\mathcal{L}} = \mathcal{L} \setminus \mathcal{L}'$. We have

$$|\bar{\mathcal{L}}| \le k(1-(2c'-3)) = k(4-2c').$$

Finally, since A is an $(r,3,c')$-expander, $|\mathrm{Vars}(\mathcal{L})| \ge c'k$. On the other hand, $|\mathrm{Vars}(\bar{\mathcal{L}})| \le 3|\bar{\mathcal{L}}| \le k(12-6c')$. Thus, the number of variables in $\mathcal{L}'$ is at least $k(c'-(12-6c')) = k(7c'-12)$.

We now apply the Gaussian elimination to the set $\mathcal{L}'$. Namely, we consequently consider variables $y \in \mathrm{Vars}(\mathcal{L}') \setminus \hat{X}$ and make substitutions $y = \ldots$ with the corresponding linear forms. It is clear that during this process every equation in (the modified) $\mathcal{L}'$ still contains at most 2 variables not from $\hat{X}$. Also, each substitution decreases the number of variables in $\mathrm{Vars}(\mathcal{L}') \setminus \hat{X}$ at most by two. Thus the Gaussian elimination ought to make at least $(k(7c'-12)-|\hat{X}|)/2$ substitutions before all variables in $\mathrm{Vars}(\mathcal{L}') \setminus \hat{X}$ are eliminated.

After this, the values of variables in $\hat{X}$ are determined by the remaining system that contains at most

$$k - \frac{k(7c'-12)-|\hat{X}|}{2} = \frac{14k-7kc'+|\hat{X}|}{2}$$

linear equations (containing only variables in $\hat{X}$); hence, the dimension of L is lower bounded by

$$|\hat{X}| - \frac{14k-7kc'+|\hat{X}|}{2} \ge |\hat{X}|\left(\frac{1}{2} - \frac{14-7c'}{2(2c'-3)}\right),$$

(here we used (3.6)). □

4 An exponential lower bound for drunk algorithms

In this section, we prove an exponential lower bound on the running time of drunk algorithms (described in Sect. 2.1) on satisfiable formulas. The proof strategy is as follows: we take a known hard unsatisfiable formula G and construct a new satisfiable formula that turns into G if the algorithm chooses a wrong value for some variable. Since for several tries the algorithms errs at least once with high probability, it follows that the recursive procedure is likely to be called on G and hence will work exponentially long.

In what follows, we give the construction of our hard satisfiable formulas (citing the construction of hard unsatisfiable formulas), then prove two (almost trivial) formal statements for the behavior of DPLL algorithms on hard unsatisfiable formulas, and, finally, prove the main result of this section.

Since the size of recursion tree for an unsatisfiable formula does not depend on the random choices of a drunk algorithm, we can assume that our algorithm has the smallest possible recursion tree for every unsatisfiable formula. We call such an algorithm an "*ideal*" drunk algorithm.

4.1 Hard satisfiable formulas based on hard unsatisfiable formulas

Our formulas are constructed from known hard unsatisfiable formulas. For example, we can take hard unsatisfiable formulas from [PI00].

Theorem 4.1 ([PI00], Theorem 1). For each $k \geq 3$ there exist a positive constant $c_k = O(k^{-1/8})$, a function $f(x) = \Omega(2^{x(1-c_k)})$ and a sequence of unsatisfiable formulas G_n in k-CNF (for each l, G_l uses exactly l variables) such that all tree-like resolution proofs of G_n have size at least $f(n)$.

Corollary 4.1. The recursion tree of the execution of a drunk DPLL algorithm on the formula G_n from Theorem 4.1 (irrespectively of the random choices made by the algorithm) has at least $f(n)$ nodes.

Proof. It is well-known that tree-like resolution proofs and DPLL trees are equivalent. Note that the subsumption rule cannot reduce the size of a DPLL tree. $\square$

Remark 4.1. We do not use other facts about these formulas; therefore, our construction works for any sequence of formulas satisfying a similar statement.

Definition 4.1. Let us fix n. We call an unsatisfiable formula F (we do *not* assume that F contains n variables) *hard* if the recursion tree of the execution of (every) "ideal" drunk algorithm on F has at least $f'(n) = (f(n) - 1)/2$ nodes, where f is the function appearing in Theorem 4.1.

Definition 4.2. We consider formulas of the form[3] $H_n = G^{(1)} \wedge G^{(2)} \wedge \cdots \wedge G^{(n)}$, where $G^{(i)}$ is the formula in CNF of n variables[4] $x_1^{(i)}, \ldots, x_n^{(i)}$ (for all $i \neq j$, the sets of variables of the formulas $G^{(i)}$ and $G^{(j)}$ are disjoint) defined as follows. Take a copy of the hard formula from Theorem 4.1; call its variables $x_j^{(i)}$ and the formula $\widetilde{G}^{(i)}$. Then change the signs of some literals in $\widetilde{G}^{(i)}$ (this is done by replacing all occurrences of a positive literal l with $\neg l$ and, simultaneously, of the negative literal $\neg l$ with l) so that the recursion tree of the execution of (every) "ideal" drunk algorithm on $\widetilde{G}^{(i)}[\neg x_j^{(i)}]$ is not smaller than that on $\widetilde{G}^{(i)}[x_j^{(i)}]$ (hence, $\widetilde{G}^{(i)}[\neg x_j^{(i)}]$ is hard). Use the (modified) formula $\widetilde{G}^{(i)}$ to construct the formula[5] $(\widetilde{G}^{(i)} \vee x_1^{(i)}) \wedge (\widetilde{G}^{(i)} \vee x_2^{(i)}) \wedge \cdots \wedge (\widetilde{G}^{(i)} \vee x_n^{(i)})$ and simplify it using the simplification rules; the obtained formula is $G^{(i)}$.

[3] Note that the subscript in H_n does *not* denote the number of variables.

[4] It is possible that some of these variables do not appear in the formula; therefore, formally, a formula is a pair: a formula and the number of its variables.

[5] We use $G \vee x$ to denote a formula in CNF: x is added to each clause of G, and the clauses containing $\neg x$ are deleted.

Remark 4.2. We change signs of literals only to simplify the proof of our result; one can think that the algorithm is actually given the input formula without the change.

Remark 4.3. It is clear that H_n has size polynomial in n (and hence in the number of variables).

4.2 Behavior of drunk algorithms on unsatisfiable formulas

Lemma 4.1. Let G be a hard formula. Let F be a formula having exactly one satisfying assigment. Let the sets of variables of F and G be disjoint. Then the formula $F \wedge G$ is hard.

Proof. The statement is easy to see (note that hardness does not depend on the number of variables in the formula): a recursion tree for the formula $F \wedge G$ correspond to a recursion tree for the formula G. □

Lemma 4.2. The formula $G^{(i)}[\neg x_j^{(i)}]$ is hard.

Proof. For each formula F by Simplify(F) we denote the result of applying the simplification rules to F (the rules are applied as long as at least one of them is applicable). It is easy to see that this formula is uniquely defined (note that our simplification rules commute with each other). By our definition of a DPLL algorithm, F is hard if and only if Simplify(F) is hard. Note that

$$\begin{aligned}&\text{Simplify}(G^{(i)}[\neg x_j^{(i)}]) = \\ &\text{Simplify}((\widetilde{G}^{(i)}[\neg x_j^{(i)}] \vee x_1^{(i)}) \wedge \cdots \wedge (\widetilde{G}^{(i)}[\neg x_j^{(i)}]) \wedge \cdots \wedge (\widetilde{G}^{(i)}[\neg x_j^{(i)}] \vee x_n^{(i)})) = \\ &\text{Simplify}(\widetilde{G}^{(i)}[\neg x_j^{(i)}]).\end{aligned}$$

(The last equality is obtained by applying the subsumption rule.) The formula Simplify($\widetilde{G}^{(i)}[\neg x_j^{(i)}]$) is hard since $\widetilde{G}^{(i)}[\neg x_j^{(i)}]$ is hard. □

4.3 Behavior of drunk algorithms on satisfiable formulas

Theorem 4.2. The size of the recursion tree of the execution of a drunk DPLL algorithm on input H_n is less than $f'(n)$ with probability at most 2^{-n}.

Proof. The unique satisfying assignment to H_n is $x_j^{(i)} = 1$. Note that $H_n[\neg x_j^{(i)}]$ contains an unsatisfiable subformula $G^{(i)}[\neg x_j^{(i)}]$.

Consider the splitting tree of our algorithm on input H_n. It has exactly one leaf corresponding to the satisfying assignment. We call node w on the path corresponding to the satisfying assignment *critical*, if Heuristic A chooses a variable $x_m^{(i)}$ for this node and this is the first time a variable from the subformula $G^{(i)}$ is chosen along this path. A *critical subtree* is the subtree corresponding to the unsatisfiable formula resulting from substituting a "wrong" value in a critical node.

By Lemmas 4.1 and 4.2 the size of a critical subtree is at least $f'(n)$ (note that the definition of a critical node implies that the corresponding subformula $G^{(i)}$ is untouched in it and hence its child contains a hard subformula $G^{(i)}[\neg x_j^{(i)}]$; it is clear that the simplification rules could not touch $G^{(i)}$ before the first assignment to its variables).

The probability of choosing the value $x_j^{(i)} = 0$ equals $\frac{1}{2}$. There are n critical nodes on the path leading to the satisfying assignment; therefore the probability that the algorithm does not go into any critical subtree equals 2^{-n}. Note that if it ever goes into a critical subtree, it has to examine all its nodes, and there are at least $f'(n)$ of them. □

Corollary 4.2. For each $k \geq 3$ there exist a positive constant $c_k = O(k^{-1/8})$, a function $g(x) = \Omega(2^{x^{(1-c_k)}})$ and a sequence of unsatisfiable formulas H_n in $(k+1)$-CNF (H_n uses m variables, where $n \leq m \leq n^2$) such that the size of recursion tree of the execution of any drunk DPLL algorithm on input H_n is less than $g(n)$ with probability at most 2^{-n}.

5 Discussion

Various generalizations of the notions of myopic and drunk algorithms would guide to natural extensions of our results. However, note that merging the notions into one is not easy: if Heuristic A is not restricted, it can feed information to Heuristic B even if it is not enabled directly (for example, it can choose variables that are to be assigned 1 while they persist). Therefore, Heuristic B must have oracle access that would hide syntactical properties of the formula so that Heuristic B would not gain any other information from Heuristic A except for "branching on the variable v is nice". For example, the oracle must randomly rename variables, (consistently) negate some of them, change the order of clauses, etc.

Acknowledgment

The authors are grateful to Eli Ben-Sasson for helpful discussions and to anonymous referees for numerous comments that improved the quality of this paper.

References

[ABM03] Dimitris Achlioptas, Paul Beame, and Michael Molloy. A sharp threshold in proof complexity. *Journal of Computer and System Sciences*, 2003.

[ABM04] D. Achlioptas, P. Beame, and M. Molloy. Exponential bounds for DPLL below the satisfiability threshold. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms, SODA04, to appear*, 2004.

[ABS02] Michael Alekhnovich and Eli Ben-Sasson. Analysis of the random walk algorithm on random 3-CNFs. Manuscript, 2002.

[ABSRW00] M. Alekhnovich, E. Ben-Sasson, A. Razborov, and A. Wigderson. Pseudorandom generators in propositional complexity. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, FOCS'00*, 2000. Journal version is to appear in *SIAM Journal on Computing.*

[AR01] M. Alekhnovich and A. Razborov. Lower bounds for the polynomial calculus: non-binomial case. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, 2001.

[AS00] Dimitris Achlioptas and Gregory B. Sorkin. Optimal myopic algorithms for random 3-SAT. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, FOCS'00*, 2000.

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

[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.

[Hir00] Edward A. Hirsch. SAT local search algorithms: Worst-case study. *Journal of Automated Reasoning*, 24(1/2):127–143, 2000. Also reprinted in "Highlights of Satisfiability Research in the Year 2000", Volume 63 in Frontiers in Artificial Intelligence and Applications, IOS Press.

[Nik02] S. I. Nikolenko. Hard satisfiable formulas for DPLL-type algorithms. *Zapiski nauchnyh seminarov POMI*, 293:139–148, 2002. English translation is to appear in *Journal of Mathematical Sciences.*

[PI00] Pavel Pudlák and Russell Impagliazzo. A lower bound for DLL algorithms for k-SAT. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'00*, 2000.

[SLBH02] Laurent Simon, Daniel Le Berre, and Edward A. Hirsch. The SAT 2002 Competition. Submitted to a journal, 2002.

[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.