ПРЕПРИНТЫ ПОМИ РАН

Учредитель: Федеральное государственное бюджетное учреждение науки
Санкт-Петербургское отделение Математического института
им. В. А. Стеклова Российской академии наук

Свидетельство о регистрации средства массовой информации: ЭЛ №ФС 77-33560 от 16 октября 2008 г. Выдано Федеральной службой по надзору в сфере связи и массовых коммуникаций

Контактные данные: 191023, г. Санкт-Петербург, наб. реки Фонтанки, дом 27

телефоны:(812)312-40-58; (812) 571-57-54

e-mail: admin@pdmi.ras.ru

http://www. pdmi.ras.ru /preprint/

Заведующая информационно-издательским сектором Симонова В.Н

PDMI PREPRINT — 01/2014

Lower Bounds for DPLL algorithms with splitting over linear functions [1]

Dmitry Sokolov

St. Petersburg Department of V.A. Steklov Institute of Mathematics
of Russian Academy of Sceinces
E-mail: sokolov.dmt@gmail.com

10 January, 2014

Abstract

A typical $DPLL$ algorithm for the Boolean satisfiability problem splits the input problem into two by assigning the two possible values to a variable; then it simplifies the resulting two formulas. There are more complicated forms of splitting (for example, splitting by a clause), but they are usually reducible to splitting over a single variable. $DPLL$ algorithms form the base of most modern SAT solvers, and there is a significant interest in exponential lower bounds for them (see, e.g., [AHI05]).

In this paper we consider an extension of the $DPLL$ paradigm. Our algorithms, $DPLL_{lin}$, can split over arbitrary linear function modulo two. These algorithms quickly solve formulas that encode linear systems modulo two, which were used for proving exponential lower bounds for conventional $DPLL$ algorithms.

We prove exponential lower bounds on the running time of $DPLL_{lin}$ algorithms. Moreover, we extend the concept of these algorithms to two newly constructed proof systems Res_{lin} and Sem_{lin} and prove exponential lower bounds on the size of tree-like proofs in these systems.

[1]The work is partially supported by Russian Foundation for Basic Research (grant mol_a 12-01-31239), president grant MK-2813.2014.1 and the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement nb.238381.

ПРЕПРИНТЫ
Санкт-Петербургского отделения
Математического института им. В. А. Стеклова
Российской академии наук

PREPRINTS
of the St. Petersburg Department of Steklov Institute of Mathematics

---

1 Introduction

$DPLL$ (named after Davis, Putnam, Logemann and Loveland) algorithms are one of the most popular approaches to the problem of satisfiability of Boolean formulas (SAT). A typical $DPLL$ algorithm is a recursive algorithm that takes the input formula ϕ, uses a procedure **A** to choose a variable x, uses a procedure **B** that chooses the value $a \in \{0, 1\}$ that will be investigated first, and makes two recursive calls for the formulas $\phi[x := a]$ the $\phi[x := 1 - a]$. Note that the second call is not necessary if the first one find the formula to be satisfiable.

There is a number of works concerning lower bounds for $DPLL$ algorithms: for unsatisfiable formulas exponential lower bounds follow from lower bounds on the complexity of resolution proofs [Urq87], [Tse68]. We have no hope to prove superpolynomial lower bound for satisfiable formulas: if P = NP, the procedure **B** may always choose the correct value of the variable according to some satisfying assignment. The paper [AHI05] gives exponential lower bounds for two wide enough classes of $DPLL$ algorithms: myopic and drunken algorithms. In the myopic case the procedures **A** and **B** can see formula with erased signs of negation, they can query the number of positive and negative occurrences for every variable and also can read $K = n^{1-\epsilon}$ clauses with negations. Drunken algorithms may have an arbitrary procedure **A**, while the procedure **B** chooses the value uniformly at random.

The paper [IS11] extends the class of $DPLL$ algorithms by adding the procedure **C** that may decide that some branch of the splitting tree will be cut (not investigated) since it is not too "perspective". More precisely, before each recursive call such an algorithm calls the procedure **C** that decides whether to make this recursive call or not. $DPLL$ algorithms with cut heuristic always give a correct answer on unsatisfiable formulas; however they may err on satisfiable formulas. On the other hand, if the presence of a cut heuristic gives a substantial improvement on the time complexity and difficult instances (i.e. instances on which the algorithm errs) are not easy to find, then such algorithms become reasonable. The paper [IS11] also proves exponential lower bounds for $DPLL$ algorithms with cut heuristic. Similarly to classical $DPLL$ algorithms the hard examples here are Tseitin formulas encoding linear systems modulo two. However, Gaussian elimination easily solves these formulas. In this work we extend $DPLL$ algorithms in order to solve linear systems modulo two by allowing to split over linear functions. We call this class of algorithms $DPLL_{lin}$.

Despite the fact that Tseitin formulas are easy for $DPLL_{lin}$ algorithms, we are still able to prove exponential lower bounds. We suggest a proof system Res_{lin} and its extension Sem_{lin} such that a fast $DPLL_{lin}$ algorithm for specific unsatisfiable formulas ϕ_n can be transformed into short proofs in tree-like Res_{lin} (and also Sem_{lin}). We prove exponential lower bounds for these systems. Our proof is based on ideas from [BPS07]. We reduce the problem to proving lower bounds on communication complexity of function $Search_\phi(x, y)$ where ϕ is an unsatisfiable formula and $Search_\phi(x, y)$ is a function, which returns number of clause which is unsatisfied by the substitution (x, y).

Open question. The following questions are still open:

- lower bounds on the dag-like version of proof systems Res_{lin} and Sem_{lin};

- lower or upper bounds on the running time of $DPLL_{lin}$ algorithms for other formulas which are hard for classical $DPLL$ algorithms (for example, the pigeonhole principle);
- lower bounds for satisfiable formulas for some bound on procedures.

2 Splitting algorithms

Let us define a linear $DPLL$ algorithm. (We denote this class of algorithms by $DPLL_{lin}$.) Such an algorithm receives a Boolean formula $\phi(x_1, x_2, \ldots, x_n)$ and an affine subspace $L \subseteq \{0,1\}^n$ and attempts to find a satisfying assignment for ϕ. We assume that the affine subspace is represented as a linear system over $\mathbb{F}_2$.

A $DPLL_{lin}$ algorithm has two procedures:

- A is a procedure that takes a formula and an affine subspace and returns a linear function over $\mathbb{F}_2$ (this is the linear function for splitting).
- B is a procedure that takes a formula, an affine subspace L and a linear function h and returns an affine subspace that will be investigated first ($L \cap (h(x) = 0)$ or $L \cap (h(x) = 1)$).

$DPLL_{lin}$ is a recursive algorithm:

Algorithm 2.1. Input: formula in CNF $\phi(x_1, x_2, \ldots, x_n)$ and affine subspace $L \subseteq \{0,1\}^n$.

1. If there exists a clause that cannot be satisfied in subspace L, then return "unsatisfiable".
2. If L contains a single point, then return this point as a satisfying assignment for formula ϕ.
3. Choose a function for splitting $h := A(\phi, L)$.
4. Choose the first value $c := B(\phi, L, h)$.
5. Make a recursive call with input $(\phi, L \cap (h = c))$. If the result is an assignment, then return it; otherwise return the result of a new recursive call with input $(\phi, L \cap (h = 1 - c))$.

Note that such an algorithm creates a tree of recursive calls. We label its internal nodes with functions used for splitting; the values are placed on the edges; the number of unsatisfied clauses (or the satisfying assignment) is placed on the leaves.

Lemma 2.1. Φ is a boolean formula in CNF with n variables of size m. We can perform the second step of algorithm 2.1 in time $p(n, m)$ for a fixed polynomial p.

Proof. There are m clauses, so we can check the compatibility of subspace L and clauses one by one.

Let us consider a clause $y_1 \vee y_2 \vee \cdots \vee y_k$, where y_i are literals. We can satisfy this clause if and only if we can satisfy one literal. $k \leq n$, so we can consider literals separately. We can satisfy literal y_k if and only if the equation $(y_k = 1)$ has a solution in space L (that is, the system $(y_k = 1) \cap L$ has a solution in $\{0,1\}^n$), and we can check it by using Gaussian elimination. □

3 $DPLL_{lin}$ Algorithms and Linear Systems

We now show that $DPLL_{lin}$ algorithms easily solve linear system over $\mathbb{F}_2$, both satisfiable and unsatisfiable ones.

We start with the following simple fact.

Proposition 3.1. If $h(x)$ is a linear function then we need at least 2^{d-1} clauses to encode it as CNF formula, where d is a number of significant variables of h.

Proof. Let X is a set of significant variables of function h and formula ϕ encodes this function. Each clause must contain all variables from set X, if not, we can set literals of this clause to zero and it implies that we set the value of formula to zero. But we substitute not all variables, so the value of formula is not a constant, contradiction.

Let us consider the substitutions which set a value of function h to 0. The number of such substitutions equals to 2^{d-1}. We match each of these substitutions to clause which turns to zero by this substitution (we always can find such clause, otherwise there is a substitution which turns formula to 1 and function to 0). If formula contains at most $2^{d-1} - 1$ clauses then for two substitutions we match one clause. This substitutions are different, so there is a variable which equals to 1 in one of substitution and equals to 0 in another substitution, thus the clause doesn't contain this variable, because it turns to zero with both substitutions, contradiction. □

Lemma 3.1. If ϕ is a formula in CNF of size m which encodes a system of linear equations $h_1(x) = c_1, h_2(x) = c_2, \ldots h_k(x) = c_k$ over field $\mathbb{F}_2$, where h_i are linear functions then there exist $DPLL_{lin}$ algorithms, such that running time of this algorithm on formula ϕ equals to $O(k + m + n)$.

Proof. We describe both procedures A and B. Without loss of generality we assume that procedures know the system of linear equations.

Procedure $A(\phi, L)$:

1. if there exists i, such that $L \cap \{x \mid h_i(x) = c_i\} = \emptyset$ and algorithm splits over equation h_i then return a significant variable of function h_i, such that algorithm hasn't split over this variable yet (if there are more than one such i choose the maximal one);

2. if there exists i, such that $L \cap \{x \mid h_i(x) = c_i\} \neq L$ then return h_i (if there are more than one such i then choose minimal one);

3. return an arbitrary linear function f such that, $L \cap \{x \mid f(x) = 0\} \neq L$ and $L \cap \{x \mid f(x) = 0\} \neq \emptyset$.

Procedure B is an arbitrary.

We now prove that this algorithm finishes after $O(k+m+n)$ steps. Let $m = \sum_{i=1}^{k} m_i$, where m_i is number of clauses which encodes an equation h_i.

If procedure A uses rule 3 then all points from current affine subspace satisfy all equation (otherwise procedure can use rule 2), so in this case we can find satisfying assignment in at most n steps (because dimension of our space is at most n).

Thus procedure A chooses equations from our system one by one. If we substitute to i-th equation the value $1 - c_i$ then procedure can use rule 1, and it chooses significant variables of this equation, because $L \cap \{x \mid h_i(x) = c_i\} = \emptyset$, and for all $j > i$ algorithm does not split over function h_j. Hence algorithm can understand that there is no satisfying assignment in this branch in 2^{d_i} steps, which is at most $2m_i$ by proposition 3.1. Thus during wrong assignments to concrete equations, algorithm makes at most $2m$ steps.

If we substitute the value c_i to i-th equation, but procedure can use rule 1 then formula is unsatisfiable (because currently we substitute only correct values to equations). If procedure starts to use rule 1 for some equation j then in 2^{d_k} steps algorithm either understands that formula is unsatisfiable or procedure A starts to use rule 1 for equation with number greater than j, thus algorithm finishes in at most $\sum_{j=1}^{k} 2^{d_j} = \sum_{i=j}^{k} 2m_j = 2m$ steps. □

4 Proof Systems

4.1 $Search_\phi$

Let ϕ is an unsatisfiable formula in CNF. We define a function $Search_\phi : \{0,1\}^n \to \{1, 2, \ldots, m\}$, where m is a number of clauses in formula ϕ. This function takes a variables substitution and returns a number of clause, which is unsatisfied by this substitution.

Definition 4.1. Decision tree for function $f : \{0,1\}^n \to \{0,1\}$ is a binary rooted tree. Every inner node is marked by some function of input variables and every leaf is marked by 0 or 1. Edges are marked by 0 or 1 and every inner node has one son with edge 0 and another with edge 1.

Calculation of function $f(x_1, \ldots, x_n)$ is started in root of tree. When we come to inner node, we calculate the function which is written in this node and move to corresponding subtree. When we come to leaf, we return the value which is written in it.

We will observe decision trees for function $Search_\phi$ such that, inner nodes are marked by linear functions. Further we will observe only such trees.

Lemma 4.1. If there exists a $DPLL_{lin}$ algorithm which works t steps on formula ϕ then there exists decision tree for function $Search_\phi$ of size at most t.

Proof. The tree of recursion calls of $DPLL_{lin}$ algorithm is a correct decision tree for function $Search_\phi$. □

4.2 Proof Systems Res_{lin} and Sem_{lin}

Let us define the proof systems Res_{lin} and Sem_{lin}. At first we define a notion of linear clause and semantic implication.

Definition 4.2. Linear clause is a clause of the following form: $(h_1(x) = c_1 \vee h_2(x) = c_2 \vee \dots h_k(x) = c_k)$, where h_i is a linear function over $\mathbb{F}_2$ and $c_i \in \{0, 1\}$.

As a negation of linear equation $h(x) = c$ we will understand an equation $h(x) = 1 - c$. There is a natural correspondence between clauses of boolean formula and linear clauses: $(x_1^{c_1} \vee x_2^{c_2} \vee \dots \vee x_k^{c_k}) \Rightarrow ((x_1 = c_1) \vee (x_2 = c_2) \vee \dots \vee x_k = c_k)$, where $x^1 = x, x^0 = \neg x$. We will call a linear clause as a clause, and respectively, instead of using clauses from boolean formulas we will use its' linear analogies.

Definition 4.3. Clause C is semantically implied by clauses $C_1, C_2, \dots, C_k$, that means that this formula is a tautology $(\bigwedge_{i=1}^{k} C_i(x)) \to C(x)$

Proposition 4.1. We can check in polynomial time if C is a semantically implied by clauses $C_1, C_2, \dots, C_k$ for fixed constant k.

Proof. Let $C = (h_1(x) = c_1 \vee h_2(x) = c_2 \vee \dots h_l(x) = c_l)$. C is not semantically implied by clauses $C_1, C_2, \dots, C_k$ if and only if for all $i \in \{1, \dots, k\}$ exists variables substitution a, such that $C_i(a) = 1$ and $C(a) = 0$.

Let us pick one equation from each clause C_i (there are at most m^k possibilities to do it, where m is a maximal size of clause), so we obtain a linear system L. And now we try to solve a linear system $L \cap (h_1(x) = c_1) \cap (h_2(x) = c_2) \cap \dots \cap (h_l(x) = c_l)$. If C is not semantically implied by other clauses then this system has solution. We can check it by using the Gaussian elimination.

Finally we use Gaussian elimination for all possibilities of creation system L. □

Definition 4.4. Proof in system Res_{lin} is a sequence of clauses $C_1, \dots, C_k$, where $C_k = \emptyset$ and C_i is created by one of following rules:

- C_i is a clause of formula;
- C_i is semantically implied by one of previous clause (analogue of weakening rule in classical resolution);
- C_i is a resolvent of two previous clauses, that means there exists $j, l < i$ such that $C_j(x) = (h_1(x) = c_1 \vee h_2(x) = c_2 \vee \dots h_s(x) = c_s)$, $C_j(x) = (h_1(x) = 1 - c_1 \vee g_2(x) = d_2 \vee \dots g_m(x) = d_m)$ and $C_i = (h_2(x) \vee h_3(x) \vee \dots \vee h_s(x) \vee g_2(x) \vee g_3(x) \vee \dots \vee g_m(x))$.

Proof system Sem_{lin} is a generalization of proof system Res_{lin}.

Definition 4.5. Proof in system Sem_{lin} is a sequence of clauses $C_1, \dots, C_k$, where $C_k = \emptyset$ and C_i is created by one of following rules:

- C_i is a clause of formula;
- C_i is semantically implied by two previous clauses.

Note that every proof in system Res_{lin} is a proof in system Sem_{lin}.

Proposition 4.2. Boolean formula has a proof in Res_{lin} if and only if this formula is unsatisfiable.

Proof. Note that the proof in resolution proof system is a proof in system Res_{lin} (it's enough to use a correspondence between boolean and linear clauses). Respectively all unsatisfiable formulas have a proof in system Res_{lin}.

Only unsatisfiable formulas have a proof in system Res_{lin}, because each clause is a clause of formula or it's semantically implied by some previous clauses. □

Proposition 4.3. Boolean formula has a proof in Sem_{lin} if and only if this formula is unsatisfiable.

Proof. It is a corollary from 4.2, because every proof in system Res_{lin} is a proof in system Sem_{lin}. □

We define a tree-like version of these systems.

Definition 4.6. Tree-like proof in system Res_{lin} (Sem_{lin}) is a correct proof, where each clause is used at most once in a creation of next clauses.

5 Lower bounds on $DPLL_{lin}$ algorithms

We will stick the to the following plan:

1. If running time of $DPLL_{lin}$ algorithm on unsatisfied boolean formula ϕ is at most s then we prove that there exists a tree-like prove of formula ϕ in system Res_{lin} of size at most $2s$.

2. If there exists a proof in tree-like proof system Res_{lin} (Sem_{lin}) of formula ϕ of size s then we create a communication protocol with depth $O(\log(\log(s))\log(s))$ to function $Search_{\phi}$.

3. We create a formula ϕ_0 for which lower bound on communication complexity on task $Search_{\phi_0}$ is known.

5.1 The connection between $DPLL_{lin}$ algorithms and proof systems Res_{lin} and Sem_{lin}

Lemma 5.1. If the running time of $DPLL_{lin}$ algorithm equals to S then there exists a proof in tree-like Res_{lin} system of the size at most $2S$.

Proof. Let us observe the recursive calls tree of $DPLL_{lin}$ algorithm. We correspond to each node v of this tree a clause c_v which contains a disjunction of negation of substitutions from path which connects root of tree and node v. Each substitution is a linear equation (thus a negation too), so c_v is a correct linear clause.

Let us observe a node v which is not a leaf and it's children v_1, v_2. If the function h_v is a splitting function in vertex v then without loss of generality $c_{v_1} = (c_v \vee (h_v = 0))$ and

$c_{v_2} = (c_v \vee (h_v = 1))$, thus c_v can be obtained from clauses c_{v_1} and c_{v_2} by using resolution rule. Also note that clause which corresponds to root, is empty, thus we obtain a correct proof from clauses which are corresponded to leafs, and the size of this proof equals to S.

To finish this proof we need to obtain the clauses which are corresponded to leafs from clauses of formula. We prove that every clause in leafs can be obtained by using one weakening rule from clause of formula.

Let us observe a leaf l. It's a leaf thus there exists a clause of formula C which is incompatible with substitution from root to leaf l. So negation of clause c_l is semantically implied by clause C.

This we can obtain all clauses from leafs by using at most S weakening rule. □

5.2 Communication complexity

Let us consider the second part of our plan. We need to create a communication protocol ([KN97]) for function $Search_\phi$ by using proof of formula ϕ in Sem_{lin} proof system (we also can use a proof in Res_{lin} proof system).

Let Alice and Bob want to solve a problem $Search_\phi$ for some boolean formula $\phi(X, Y)$. Alice knows values of variables in X and Bob knows values of variables in Y.

We denote by $R^{pub}_\epsilon(f(X, Y))$ a minimal number of bits which Alice and Bob need to communicate to calculate a value of function f with probability at least $1 - \epsilon$. They use common random bits (Alice knows values of X and Bob knows values of Y), ([KN97], ch. 3).

Lemma 5.2. For all $\epsilon = \epsilon(n)$ and all linear clause $c(x_1, \ldots, x_n, y_1, \ldots, y_n)$. $R^{pub}_\epsilon(c) \leq O(\log(\frac{1}{\epsilon}))$.

Proof. Let $c = (h_1(x, y) = c_1 \vee h_2(x, y) = c_2 \vee \cdots \vee h_k(x, y) = c_k)$. Let us describe Alice's and Bob's strategies.

1. Choose a random subset of linear equations I from clause c.
2. Bob sends to Alice a bit $v = \sum_{i \in I} h_i(0, y)$.
3. Alice calculates a bit $v' = \sum_{i \in I} h_i(x, 0) + v$.
4. If $v' = \sum_{i \in I}(c_i + 1)$ then the answer is 0 else answer is 1.
5. Alice sends the answer to Bob.

Communication complexity of this protocol equals to 2. Let us prove that if answer is 0 then protocol always returns 0, in the other case protocol returns 1 with probability $\frac{1}{2}$.

We observe a bit v'. $v' = \sum_{i \in I} h_i(x, 0) + v = \sum_{i \in I} h_i(x, 0) + \sum_{i \in I} h_i(0, y) = \sum_{i \in I} h_i(x, y)$. If all equations in clause are false then $h_i(x, y) = c_i + 1$, hence $v' = \sum_{i \in I}(c_i + 1)$, hence in this case protocol is correct. If subset A of equations is true then with probability $\frac{1}{2}$ the

following equality is true $|I \cap A| = 1 \ mod\ 2$. Thus with probability $\frac{1}{2}$ next equality is true: $v' = \sum_{i \in I} h_i(x,y) = \sum_{i \in I \cap A} c_i(x,y) + \sum_{i \in I \setminus A} (c_i(x,y)+1) \neq \sum_{i \in I}(c_i+1)$.

To obtain the error at most ϵ it is enough to repeat this protocol $O(\log(\frac{1}{\epsilon}))$ times. □

We need the following lemma to create the protocol.

Lemma 5.3. If T is a tree and each node has at most two children then there exists a node $v_0 \in T$, such that the size of subtree with root v_0 is at least $\frac{1}{3}$ and at most $\frac{2}{3}$ of number of nodes in T.

Proof. Let us denote by s the size of tree T and by T_v a subtree with root v. We describe an algorithm to find a vertex v_0.

Input: a node v and a tree T of size s.

1. If the size of T_v is at least $\frac{1}{3}s$ and at most $\frac{2}{3}s$ then return a node v.
2. Choose a child v_1 of the node v such that subtree of this child is maximal.
3. Recursive call with input (v_1, T).

We run this algorithm for tree T and the root of tree T.

If v is in T and the size of T_v is at least $\frac{2}{3}s$ then the size of T_{v_1} is at least $\frac{1}{3}s$. Thus algorithm either stops and return a correct vertex or the size of current subtree is always bigger than $\frac{2}{3}s$ but after at most s step we come to a leaf where this size is equals to 1. □

Lemma 5.4. If there exists a tree-like proof of formula ϕ in Sem_{lin} proof system then $\forall \epsilon > 0 \ \ R^{pub}_{\epsilon}(Search_{\phi}) \leq O((\log(\frac{1}{\epsilon}) + \log\log(S))\log(S))$.

Proof. We denote by T a tree, by $|T|$ the size of tree T and by T_v a subtree with root v. v_{mid} is a node such that $\frac{1}{3}|T| \leq |T_{v_{mid}}| \leq \frac{2}{3}|T|$ (if the tree T is a binary tree then this node always exists by lemma 5.3). We correspond to each node v the clause c_v.

Let's describe Alice's and Bob's strategies. Protocol will be recursive. Alice has values for the variables in set X and Bob has values for the variables in set X. T is a subtree of proof in tree-like Sem_{lin} proof system. Some branches can be removed from subtree T.

1. If T contains only one node then return clause which corresponds to this node.
2. Find a node v_{mid} in tree T.
3. Use a lemma 5.2 to calculate the value of clause $c_{v_{mid}}$ with probability $\frac{\epsilon}{\log_{\frac{3}{2}}(S)}$.
4. If clause is true then make a recursive call with input $(X, Y, T \setminus T_{v_{mid}})$.
5. If clause is true then make a recursive call with input $(X, Y, T_{v_{mid}})$.

At first we observe a communication complexity of this protocol. Bits are transmitted only on step 3 and the number of bits is at most $\log(\frac{1}{\epsilon}) + \log\log(S)$ by lemma 5.2. At each recursive call the the size of tree is divided by at least $\frac{3}{2}$ (by choosing a node v_{mid}),

thus there are $O(\log(S))$ recursive steps, thus communication complexity of this protocol equals to $O((\log(\epsilon) + \log\log(S))\log(S))$.

Now we observe an error probability. Let's assume that step 3 of our protocol is always correct. Note that we have invariant: root of current tree contains a clause which is not satisfied by Alice's and Bob's values (at first we run a protocol for whole tree which has an empty clause in the root). Thus protocol always returns an unsatisfied clause and we need to show that this clause is contained in our formula.

If our protocol returns clause c_v which is not contained in formula (not in the leaf of our tree) then both children were cut on step 4, thus clauses in children are satisfied by substitution. Clause c_v is semantically implied by the children, thus c_v must be satisfied, contradiction.

The probability of error on step 3 is at most probability from lemma 5.2 multiply by number of recursive calls, thus this probability is at most $\frac{\epsilon}{2\log_{\frac{3}{2}}(S)}\log_{\frac{3}{2}}(S) \leq \epsilon$. □

5.3 Lower bounds on communication protocol

Definition 5.1. Tseitin formula is based on a simple connected undirected graph $G = (V, E)$ with degree bounded by a constant d. Every edge $e \in E$ has the corresponding propositional variable x_e, every vertex $v \in V$ has the corresponding constant $c_v \in \{0, 1\}$. For every vertex we write a formula in CNF that codes an equality $\sum\limits_{u,(v,u)\in E} x_{(v,u)} = c_v$.

We denote this formulas as a $TS_{(G,c)}(x)$.

We now modify the Tseitin formulas. k-fold Tseitin formulas ([BPS07]) are based on Tseitin formulas. We receive a Tseitin formula $TS_{(G,c)}$ and change each variable x_i to the following formula $(z_{i1} \wedge z_{i2} \wedge \cdots \wedge z_{ik})$, after we translate resulting formula to CNF and we receive formula $TS^k_{(G,c)}(z)$.

Note that if the degree of graph G is bounded then for every natural constant k the size of $TS^k_{(G,c)}$ is bounded by some polynomial in number of vertexes in G.

We observe the 2-fold Tseitin formulas and we assume that Alice has values for variables z_{i1} and Bob has values for variables z_{i2}.

Theorem 5.1. There exists an explicit family of graphs G_n and vectors c_n such that $R^{pub}_{\frac{1}{3}}(Search_{TS^2_{(G,c)}}) = \Omega(\frac{n^{\frac{1}{3}}}{(\log(n)\log\log(n))^2})$.

We will use a results of work [BPS07] to prove this theorem. The main idea is to reduce the problem $DISJ_{n,k}$ to our problem with $k = 2$.

Definition 5.2. Let $A, B \subseteq \{1, \ldots, n\}$ then $DISJ_{n,2}(A, B) = 1 \Leftrightarrow A \cap B = \emptyset$.

Theorem 5.2. ([BPS07] ch. 5) Let $m = \frac{n^{\frac{1}{3}}}{\log(n)}$ then for every n there exists a vector $c \in \{0, 1\}^n$ and an explicit graph G with n vertexes such that
$R^{pub}_{\epsilon}(DISJ_{m,2}) = O(R^{pub}_{\epsilon}(Search_{TS_{(G,c)}})\log(n)(\log\log(n))^2)$.

Lemma 5.5. ([KS92]) $R^{pub}_{\frac{1}{3}}(DISJ_{n,2}) = \Omega(n)$.

We are ready to prove Theorem 5.1.

Proof. (Theorem 5.1)

Let $m = \frac{n^{\frac{1}{3}}}{\log(n)}$. By lemma 5.5, $R^{pub}_{\epsilon}(DISJ_{m,2}) = \Omega(m)$ thus by Theorem 5.2 there exists G, c such that $R^{pub}_{\frac{1}{3}}(Search_{TS^2_{(G,c)}}) = \Omega(\frac{n^{\frac{1}{3}}}{(\log(n)\log\log(n))^2})$ □

5.4 Lower Bounds on Sem_{lin} and Res_{lin}

Theorem 5.3. There exists an explicit family of graphs G_n with n vertexes and vectors c_n such that, the size of proof of formula $TS^2_{(G_n,c_n)}$ in tree-like proof systems Sem_{lin} and Res_{lin} equals to $\Omega(2^{n^{\frac{1}{3}}/\log^3(n)})$.

Proof. Let us use a family from Theorem 5.1.
$R^{pub}_{\frac{1}{3}}(TS^2_{(G,c)}) = \Omega(\frac{n^{\frac{1}{3}}}{(\log(n)\log\log(n))^2})$, but by lemma 5.4
$R^{pub}_{\epsilon}(Search_{\phi}) \leq O((\log(\epsilon) + \log\log(S))\log(S))$ where S is the proof size thus $\log\log(S)\log(S) = \Omega(\frac{n^{\frac{1}{3}}}{(\log(n)\log\log(n))^2})$ thus $S = \Omega(2^{n^{\frac{1}{3}}/\log^3(n)})$ □

Corollary 5.1. There exists an explicit family of graphs G_n with n vertexes and vectors c_n such that, the running time of each $DPLL_{lin}$ algorithm on formulas from family $TS^2_{(G_n,c_n)}$ equals to $\Omega(2^{n^{\frac{1}{3}}/\log^3(n)})$.

Proof. It is a corollary from Theorem 5.3 and lemma 5.1. □

Acknowledgements. The author is grateful to Jan Krajíček, to Dmitry Itsykson and to Edward A. Hirsch for fruitful discussions and error correction.

References

[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.

[BPS07] Paul Beame, Toniann Pitassi, and Nathan Segerlind. Lower bounds for lovász-schrijver systems and beyond follow from multiparty communication complexity. *SIAM Journal on Computing*, 37(3):845–869, 2007.

[IS11] Dmitry Itsykson and Dmitry Sokolov. Lower bounds for myopic DPLL algorithms with a cut heuristic. In *Proceedings of the 22nd international conference on Algorithms and Computation*, ISAAC'11, pages 464–473, Berlin, Heidelberg, 2011. Springer-Verlag.

[KN97] Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, New York, NY, USA, 1997.

[KS92] Bala Kalyanasundaram and Georg Schintger. The probabilistic communication complexity of set intersection. *SIAM J. Discret. Math.*, 5(4):545–557, November 1992.

[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.